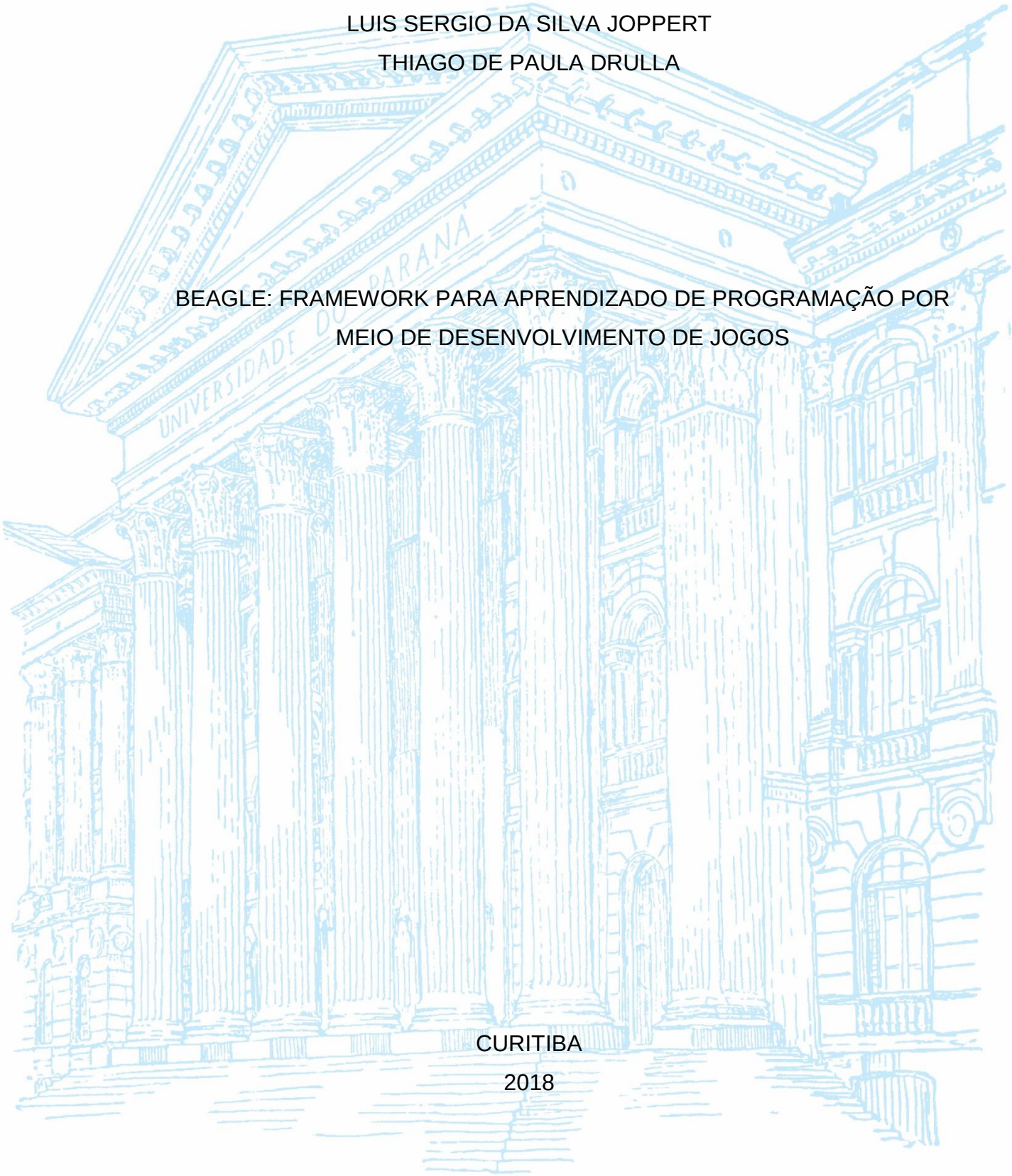
UNIVERSIDADE FEDERAL DO PARANÁ

GABRIEL VIEIRA DE QUEIROZ
LUCAS ALVES COUTINHO GEHLEN
LUIS SERGIO DA SILVA JOPPERT
THIAGO DE PAULA DRULLA

BEAGLE: FRAMEWORK PARA APRENDIZADO DE PROGRAMAÇÃO POR MEIO DE DESENVOLVIMENTO DE JOGOS



CURITIBA
2018

GABRIEL VIEIRA DE QUEIROZ
LUCAS ALVES COUTINHO GEHLEN
LUIS SERGIO DA SILVA JOPPERT
THIAGO DE PAULA DRULLA

BEAGLE: FRAMEWORK PARA APRENDIZADO DE PROGRAMAÇÃO POR MEIO DE DESENVOLVIMENTO DE JOGOS

Trabalho de conclusão de curso apresentado ao curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientadora: Profa. Msc. Andreia de Jesus

CURITIBA
2018

TERMO DE APROVAÇÃO

Gabriel Vieira de Queiroz

Lucas Alves Coutinho Gehlen

Luis Sergio da Silva Joppert

Thiago de Paula Drulla

B.E.A.G.L.E. - Biblioteca Educacional para Aprendizado Gráfico de Lógica em Escolas

Monografia aprovada como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Setor de Educação Profissional e Tecnológica da Universidade Federal do Paraná.

Andreia de Jesus

Profª. Andreia de Jesus

Orientadora – SEPT/UFPR

Profº. Luiz Antonio Pereira Neves

SEPT/UFPR

Jouglas Alves Tomaschitz

Profº. Jouglas Tomaschitz

SEPT/UFPR

Curitiba, 05 de Dezembro de 2018.

Dedicamos este trabalho,

Aos nossos pais, Lenira da Silva, Aparecida Vieira, Adir Drulla, Angela Drulla, Valdir Gehlen e Michelly Gehlen, por todo o carinho, por pavimentarem o caminho para o sucesso e por tudo o que fizeram e fazem por nós.

A Fernanda Carvalho da Silva pelo carinho, paciência e dedicação ao estar do lado e dar forças para seguir em frente (em nome de Lucas Gehlen).

A Gabriel Aquino e Lucas Canestraro pelo apoio e carinho apoiando esta etapa de nossas vidas (em nome de Gabriel Vieira e Lucas Gehlen).

A Daniel Henrique Sepulveda Hirooka por toda a dedicação ao desenvolvimento dessa ferramenta antes de seu afastamento.

AGRADECIMENTOS

À Universidade Federal do Paraná - UFPR, na pessoa do Reitor Prof. Dr. Ricardo Marcelo Fonseca.

Ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, na pessoa do Coordenador Prof. João Eugenio Marynowski.

Aos docentes do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas.

À Profa. Andreia de Jesus, pela dedicação, amizade e paciência em nos orientar na elaboração deste trabalho.

À Profa. Rafaela Montovani Fontana, pelo convívio fraterno e apoio ao desenvolver este trabalho.

À Prof. Luiz Antônio Pereira pela amizade e apoio no desenvolvimento deste trabalho.

À Daniel Henrique Sepulveda Hirooka por toda a ajuda e emprenho no desenvolvimento deste trabalho.

“I may not have gone where I intended to go, but I think I have ended up where I needed to be”. (ADAMS, 1988)

RESUMO

Este trabalho tem como premissa auxiliar iniciantes na área de programação a começarem a implementar jogos, por meio de um framework intuitivo e transparente. Visto que o desenvolvimento de jogos pode ser uma alternativa mais interessante para quem ainda está começando a programar, além de trabalhar algumas áreas pouco vistas na concepção de softwares convencionais, como por exemplo, física, balística e colisão. O framework é concebido na plataforma *Processing*, implementado na linguagem de programação Java e denominado B.E.A.G.L.E - Biblioteca Educacional para Aprendizado Gráfico de Lógica em Escolas. Esta ferramenta baseia-se em funções simples para auxiliar na construção de um jogo usando poucas linhas de código, considerando que, atualmente, as alternativas de bibliotecas gráficas comumente utilizadas no ensino da programação são complexas de se entender e limitadas em muitos aspectos. O B.E.A.G.L.E oferece várias funções para o desenvolvimento de jogos em física, animação e outras áreas. Alguns exemplos de funções são: criar um objeto estático ou animado, rotacionar objeto, ativar e desativar sons, ativar modo desenvolvedor, movimentar objeto e outras funções. Essas funções têm como objetivo serem fáceis de usar e transparentes para o desenvolvedor.

Palavras-chave: Jogos Digitais. Desenvolvimento. Lógica de Programação. Aprendizado. *Processing*.

ABSTRACT

This project has as premise to help beginners in the programming area to begin to implement games, through an intuitive and transparent framework. Since the development of games can be a more interesting alternative for those who are still starting to program, in addition to working on some areas not seen in the design of conventional software, such as physics, ballistics and collision. The framework is designed in the Processing platform, implemented in the Java programming language and called B.E.A.G.L.E - Educational Library for Logical Learning in Schools. This tool is based on simple functions to aid in the construction of a game using a few lines of code, considering that, currently, the alternatives of graphical libraries commonly used in programming teaching are complex to understand and limited in many aspects. B.E.A.G.L.E offers several functions for the development of games in physics, animation and other areas. Some examples of functions are: create a static or animated object, rotate object, activate and deactivate sounds, activate developer mode, move object and other functions. These functions are intended to be easy to use and transparent to developers.

Keywords: Digital Games. Development. Programming Logic. Learning. Processing.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS OU SIGLAS

SGBD - Sistema Gerenciador de Banco de Dados

PHP - “Hypertext PreProcessor”

OO - Orientação a Objetos

TADS - Tecnologia em Análise e Desenvolvimento de Sistemas (curso)

MVC - Padrão “Model View and Controller”

B.E.A.G.L.E - Biblioteca Educacional para Aprendizado Gráfico de Lógica em Escolas

IDE - Ambiente de desenvolvimento integrado (“Integrated Development Environment”)

LISTA DE SÍMBOLOS

© - copyright

@ - arroba

® - marca registrada

SUMÁRIO

1 INTRODUÇÃO

Na atualidade é imenso o horizonte de ferramentas tecnológicas que auxiliam na execução de tarefas de diversas áreas de conhecimento. E essas ferramentas estão sendo criadas e melhoradas cada vez mais rápido, seja hardware ou software. E, com essa inovação da tecnologia vem à tona o ápice das descobertas e da re-padronagem, com métodos de se desenvolver tecnologia de maneira mais rápida e conveniente.

É importante colocar aqui que o uso dessas ferramentas pode ir além da execução de atividades repetitivas, elas podem ser aplicadas em atividades que visam o desenvolvimento e a aprendizagem de novos conhecimentos. Como a integração das ferramentas tecnológicas com a área da educação que pode melhorar processos de aprendizagem iniciais/introdutórios existentes nas diversas modalidades de ensino (fundamental, médio, técnico, superior, pós-graduação). O leque de oportunidades na área de tecnologias educacionais é vasto e é justificado pelos diversos softwares, sejam para auxílio educacional, jogos de desenvolvimento de raciocínio lógico, ferramentas que contribuam com o desenvolvimento intelectual e até mesmo ferramentas que encapsulam conceitos de programação e que facilitam o desenvolvimento e a compreensão do raciocínio computacional. A exemplo disso, tem-se os frameworks, também de grande relevância na área de Tecnologia da Informação (TI), pois permitem que os conhecimentos de diferentes desenvolvedores sejam agrupados em um único framework, que quando disponibilizado à comunidade torna possível que uma nova estrutura de desenvolvimento de aplicações seja disseminada.

Logo, na contextualização entre as áreas de TI e Educação se destaca a semelhança de sempre haver a busca pelo aprimoramento do processo de desenvolvimento; que para a área de TI é a busca de um framework que permita um desenvolvimento facilitado com a persistência da qualidade; enquanto que para a área de ensino se busca métodos de desenvolvimento que busquem facilitar da melhor maneira possível a aprendizagem do aluno.

Sendo assim, os frameworks podem ser contextualizados na área de TI, segundo Medeiros (2014), como agregadores de funções ou funcionalidades básicas e comuns para várias aplicações. Diversos são os exemplos de frameworks, seja para um tipo específico de aplicação em determinada área e/ou problema ou para uma

tecnologia específica. Outra condição imposta a eles é a característica de serem pedaços de códigos repartidos em conjuntos, que por sua vez podem ser utilizados de acordo com as necessidades da aplicação a ser desenvolvida.

Então, voltando a integração entre TI e Educação, destaca-se o uso de ferramentas como frameworks para o desenvolvimento de atividades de ensino, que em sua maioria disponibiliza interfaces agradáveis, proporcionando para o aluno mais uma forma de compreender conceitos novos e abstratos, tudo isso sem a necessidade de haver um conhecimento do funcionamento interno da ferramenta.

1.1 PROBLEMA

Considerando o campo da aprendizagem de programação de computadores, mais especificamente no desenvolvimento de softwares computacionais gráficas, seja para jovens do ensino fundamental/médio ou para iniciantes da área de TI que estão começando a vida acadêmica universitária, existe uma barreira de abstração de código computacional em algo visual.

Se levarmos em consideração um jogo, a parte gráfica muitas vezes é um aspecto muito importante para o usuário, porém a não ser que seja um software exclusivamente feito para jogos (sendo muito desses limitados apenas as ferramentas e interfaces que são oferecidas pelos mesmos) gerar gráficos parar um jogo pode vir a ser uma tarefa muito complicada.

Pensando em bibliotecas gráficas, um exemplo é o GTK que é um conjunto de bibliotecas desenvolvido para GIMP[1] que é o *software* voltado para artes gráficas mais utilizado na plataforma GNU/LINUX, sendo a linguagem perfeita para o desenvolvimento de aplicações gráficas livres ou comerciais (GTK-PHP, Online, 2018). No mesmo seguimento existe o QTDESIGNER que usando os mecanismos de sinais e *slots* da Qt[2] permite a construção de interfaces gráficas com foco no usuário possuindo uma grande facilidade de integração junto a ambientes programados (QTDESIGNER, Online, 2018).

---

[1] GIMP é um programa de código aberto voltado principalmente para criação e edição de imagens raster, e em menor escala também para desenho vetorial.
[2] QT é a desenvolvedora da biblioteca QTDESIGNER.

As ferramentas anteriormente citadas são voltadas ao desenvolvimento de interfaces, entretanto um jogo vai além das interfaces, precisando muitas vezes de personagens e animações, objetos que essas bibliotecas não oferecem. Entretanto o B.E.A.G.L.E oferece tais ferramentas na forma de uma função simples, além de executar em cima do software Processing, esse que já tem suporte gráfico nativo.

Agora pensando em ferramentas disponíveis no mercado para o ensino de programação de computadores, podemos citar, entre outras, o Scratch e o Code.org. Segundo o CODESPEAKLABS (2017), essas ferramentas têm as seguintes características e limitações: a ferramenta Code.Org tem como principais diferenciais a preocupação com funcionalidades destinadas à professores, tendo por isso grande apoio dos mesmos; permite que o aluno gere o código em uma linguagem real, o JavaScript; também possui uma boa compatibilidade mobile. No entanto, um de seus principais pontos negativos é a falta de liberdade de desenvolvimento o que, com o tempo, acaba levando ao desinteresse por parte dos alunos. Já o *Scratch*, tem como principal destaque a liberdade permitida ao aluno, porém, ao contrário do *Code.Org*, não realiza a geração de um código real, além de possuir poucas ferramentas para professores, o que acaba reduzindo o uso da ferramenta por esse grupo.

No Brasil, o desenvolvimento de jogos vem crescendo, não ficando restrito somente a diversão, mas também atingindo a área educacional. Conforme coloca MELO (2012, p.22), para fins pedagógicos, o MEC adotou o Linux Educacional, um Sistema Operacional de código aberto que disponibiliza diversos softwares, incluindo os educacionais. Um exemplo é o GCompris, que disponibiliza diversos mini games como: Mina de Ouro, A história de Louis Braille, Chronos entre outros que além de ajudar no desenvolvimento educacional da criança pode também auxiliar na fixação de conteúdos abordados em sala aula.

Com a inclusão de ferramentas dessa natureza (Scratch, Code.org, GCompris) na área educacional, vem por despertar a curiosidade de crianças e adolescentes de desenvolver o seu próprio jogo, tornando-se produtores e não somente consumidores de jogos. Logo, diante do que foi colocado, surge a seguinte questão: o desenvolvimento de jogos pode facilitar o aprendizado da lógica e programação de computadores?

1.2 JUSTIFICATIVA

Segundo Picanço (2016), o processo de aprendizado de programação de computadores e lógica computacional é, em geral, complicado para alunos principiantes. Conforme Savi (2011 apud PICANÇO, 2016), a utilização de métodos alternativos para o ensino da programação seria uma forma de minimizar a complexidade para abstrair os conceitos. Exemplos de alternativas seriam estudos de caso, atividades realizadas em empresas da área, jogos e outros mais. Ainda de acordo com Picanço, utilizar-se de frameworks para auxiliar no ensino de programação funciona como contrapartida a falta de experimentos práticos que comumente ocorre em aulas de linguagem de programação.

Sendo assim, propõe-se a implementação do framework B.E.A.G.L.E para o ensino de programação e lógica de computadores. Trata-se de um framework para desenvolvimento de jogos.

O primeiro ponto para a aplicação do B.E.A.G.L.E. é que ele é construído em "cima" do *Processing*, um software flexível para codificar em um contexto de artes visuais (*Processing*, Online, 2018, tradução nossa[3]). Isto já abre um leque de alternativas para quem é iniciante e deseja se aventurar dentro da plataforma. Pelo fato do *Processing* ser flexível, o usuário pode instalar extensões para várias linguagens de programação diferentes, sendo assim, deu a oportunidade de codificar o B.E.A.G.L.E. em Java. O fato de codificado em Java atribui o segundo ponto para se aplicar o framework B.E.A.G.L.E., visto que Java é uma linguagem de programação amplamente usada no mundo todo, e sua plataforma está presente em mais de 3 bilhões de dispositivos (Java, Online, 2018). Por fim, o B.E.A.G.L.E. é um framework para o desenvolvimento de jogos, um tema que pode vir a ser bem mais atrativo que os exercícios ou temas clássicos geralmente abordados em aula de programação.

Levando em conta as considerações anteriores, utilizar um framework, neste caso o B.E.A.G.L.E., pode acrescentar muito no aprendizado de um aluno. A utilização do framework facilita o desenvolvimento das primeiras aplicações, visto que o estudante utilizará, em suma, as funções já feitas pelo framework. Isto poupa o

---

[3] "Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts".

desenvolvedor, à primeira vista, do código pesado, podendo reduzir o estresse ao codificar, visto que possibilita a um programador amador desenvolver aplicações com alto grau de complexidades usando funções simples e intuitivas.

Ademais, Maia (2010 apud PICANÇO, 2016) aponta que as vantagens de um framework estão na redução da quantidade de linhas a serem escritas; na oportunidade de usar juntamente técnicas alternativas de reuso, como por exemplo, padrões de projeto, redução de erros, aumento da qualidade e simplicidade na manutenção. Além das vantagens citadas as funções do B.E.A.G.L.E. estão padronizadas para o português brasileiro, ajudando um iniciante que não esteja familiarizado com o inglês a programar.

Portanto, espera-se que um aluno, aprendendo programação pelo B.E.A.G.L.E., tenha a oportunidade de ser inserido em uma plataforma diferente das clássicas usadas para ensino (*Processing*), programando em uma linguagem de programação bem estabelecida e amplamente usada (Java). Além disso, a temática de desenvolvimento do B.E.A.G.L.E. é jogos, um tema mais dinâmico e bem mais atrativo para o público mais jovem, além de abordar conhecimentos que nem sempre são utilizados na programação convencional, como por exemplo, aplicação prática de física (gravidade, balística, entre outros). Ao final, além do aluno poder desenvolver competência nas áreas citadas, ele terá como projeto pessoal um jogo eletrônico, que inclusive poderá ser fruto de entretenimento para o próprio aluno e para outros.

1.3 OBJETIVO GERAL

O objetivo deste trabalho acadêmico consiste no desenvolvimento de um *framework* educacional, denominado B.E.A.G.L.E. (Biblioteca Educacional para Aprendizado Gráfico de Lógica em Escolas), para auxiliar a aprendizagem de lógica de programação por meio do desenvolvimento de jogos.

1.4 OBJETIVOS ESPECÍFICOS

- Pesquisar e selecionar uma metodologia de desenvolvimento de *framework*.

- Implementar conceitos de processamento de imagens aplicado a jogos, colisão e posicionamento, e também recursos básicos da física para jogos.
- Modelar, documentar e desenvolver o *framework* **B.E.A.G.L.E.**
- Desenvolver um protótipo de jogo simples, utilizando o *framework* **B.E.A.G.L.E.**, a fim de demonstrar a sua usabilidade.

Nos próximos capítulos serão apresentadas as vantagens que a tecnologia pode trazer para educação além também de explicar os conceitos de jogos digitais que é um dos pilares centrais da B.E.A.G.L.E., além disso, será apresentado as características de um framework e também de suas possíveis classificações. Por fim, será apresentado como foi realizado processo e também a metodologia de desenvolvimento utilizada no framework B.E.A.G.L.E. além de contextualizar também o funcionamento do jogo exemplo criado para pôr em prática as funcionalidades do framework.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados os conceitos que são aplicados na fundamentação da ferramenta, bem como os conceitos aplicados no desenvolvimento do *framework* e os trabalhos similares, com o propósito de embasar o domínio que será abordado neste trabalho.

2.1 TECNOLOGIAS EDUCACIONAIS E O ENSINO DE LINGUAGEM DE PROGRAMAÇÃO

O ensino, em todas as atmosferas, vem evoluindo há muito tempo, desde a Grécia antiga até os dias atuais. Entretanto, com a Tecnologia da Informação (TI) a educação começou a evoluir com muito mais velocidade, visto que com ela vieram muitas ferramentas que auxiliam no aprendizado.

Todavia percebeu-se que a tecnologia poderia ser aplicada ao ensino, melhorando a forma como o conhecimento é agregado ao aluno. Tais ferramentas se englobam dentro das Tecnologias Educacionais. De acordo com Gerbran (2009), a tecnologia educacional é um meio e não um fim para o ensino, já que a mesma é responsável por conectar o aluno, o professor e a experiência pedagógica. Porém, tal nomenclatura é usada de forma errônea, visto que a tecnologia é apenas um auxílio à educação e não o foco principal. Uma forma pertinente de nomear seria “Educação Mediada pela Tecnologia”.

Indo para uma área de nicho, as Tecnologias Educacionais, podem e são usadas no ensino de programação em escolas e faculdades. Funciona como uma metalinguagem onde o programa desenvolvido pela programação ensina outros a programar. Com isso Rapkiewicz (2005, p. 2353) aborda quatro problemas principais no ensino da programação: (1) o primeiro é a dificuldade de interpretar o problema, o que impacta no desenvolvimento da resolução; (2) o segundo é a dificuldade em conseguir extrair os dados necessários para o desenvolvimento do algoritmo; (3) já o terceiro problema é a dificuldade em empregar habilidades conquistadas anteriormente no desenvolvimento; por fim, (4) o quarto e último problema listado é a primordialidade em definir um paradigma da programação na resolução de problemas, já que isto em alguns casos pode fechar o aluno para outras eventuais soluções por meio de outras linguagens.

Levando em consideração os problemas citados é possível aplicar algumas ferramentas já conhecidas para eventualmente ajudar a amenizar tais particularidades. Com relação a interpretação de problemas o Code.org tem vários exercícios simples para poder treinar interpretação, inclusive dando dicas para a resolução, caso o aluno comece a ter dificuldades. Ainda se tratando do Code.org, considerando o segundo problema, os exercícios são dispostos, além da forma escrita convencional, de uma forma gráfica, com isso incita o aluno de forma lúdica a sempre analisar mais de uma perspectiva de solução para o problema.

Já dentro do terceiro problema a estrutura do Code.org também pode auxiliar, pois ela ensina funcionalidade por funcionalidade como a programação funciona e, com isso, vai cada vez mais incrementando os exercícios com o que foi aprendido, assim o aluno sempre pratica o que já foi aprendido anteriormente. Por fim, para o quarto problema, nem o Code.org nem o *Scracth* podem oferecer um suporte tão bom no início do aprendizado da programação, já que paradigmas de programação vai de encontro a linguagens de programação e suas particularidades, vantagens e desvantagens. Porém, ao realizar vários exercícios nessas plataformas existe a abertura de novos horizontes, visto que o aluno não estará incluso em um paradigma propriamente dito.

Não obstante, a utilização de Tecnologias Educacionais fica perceptível no auxílio do aprendizado, não apenas nas matérias básicas do ensino fundamental e médio, mas até em matérias complexas na graduação. Com isso, se tratando do ensino da programação o uso de tais ferramentas não só melhora a abstração do conhecimento como é muito mais atrativo, principalmente para crianças. Baseado nisso, Joppert et. al (2017) mostra como utilizar algumas Tecnologias Educacionais para o ensino da programação dentro do projeto de extensão Construindo Saberes Através do Computador e Internet – 2a Edição, executado pela Universidade Federal do Paraná, na oficina Raciocinando com o Computador.

No projeto foram utilizadas duas das ferramentas já citadas para ensinar programação. Em primeira instância foi utilizado o *Code.Org* para ensinar os conceitos básicos de programação, como por exemplo, condicionais laços e programação estruturada. Neste caso o avanço dos alunos foi de forma igualitária, com pouca diferença na evolução nos exercícios para cada aluno. Já para o trabalho final da oficina foi utilizado o *Scracth*, abordando todos os conteúdos ensinados para o desenvolvimento de um jogo simples.

Contudo, analisando de forma ampla, já é perceptível que a metodologia implementada no projeto de extensão auxiliou nos três primeiros obstáculos citados por Rapkiewicz anteriormente, já que todos os alunos conseguiram finalizar o trabalho utilizando, em suma, apenas as diretrizes das ferramentas propostas, precisando de pouca orientação dos professores.

Portanto, a utilização de ferramentas educacionais pode ser de grande importância no ensino, pois a partir dela é possível que o aluno faça a abstração do conteúdo proposto de forma lúdica, auxiliando muitas vezes em problemas graves de aprendizado que podem prejudicar o aluno em toda a sua vida acadêmica e, dependo do caso, como na área de TI, pode até mesmo prejudicar na vida profissional.

2.2 O QUE SÃO JOGOS DIGITAIS?

De acordo com SCHUYTEMA (2008), os jogos eletrônicos se caracterizam por possuir ações e decisões que resultam em uma condição final. Essas, por sua vez, são limitadas por regras junto ao universo do *game*, que no caso dos jogos digitais são controlados por algoritmos computacionais de diferentes linguagens, como Java, *Processing* , C++ e etc. O universo tem como função contextualizar as ações praticadas pelo jogador, utilizando para isso a ambientação e narrativa; as regras têm como objetivo delimitar as ações que o jogador pode realizar, estando por conta disso diretamente ligado aos diferentes níveis de dificuldade, pois delimita o que será necessário para o jogador alcançar determinado objetivo.

E, também, segundo BATTAIOLA (2000), os jogos eletrônicos são compostos por três pilares: enredo, motor e interface interativa. O enredo tem a responsabilidade de definir como vai ocorrer a história do jogo, como por exemplo, objetivos, personagens e sequência de acontecimentos. O motor tem a responsabilidade de realizar a ponte entre o jogador e o motor gráfico, ou seja, permitir que o jogador compreenda as ações que podem ser realizadas através de efeitos audiovisuais.

Já os jogos digitais estão intimamente ligados a computadores, consoles e celulares e, por isso, é notável, segundo LUCCHES e RIBEIRO (2009), que muitos jogos digitais são representações de jogos de forma computacional, um exemplo seriam os jogos de tabuleiro *online* que em geral seguem as mesmas regras do formato físico.

Segundo JULL (2005), o principal fator que caracteriza um jogo digital é a existência de um mundo fictício, ou seja, cada jogo possui o seu próprio universo onde ocorre todo o seu desenvolvimento, sendo esse um fator que torna um jogo único um do outro. Portanto, é possível constatar que os jogos digitais estão fortemente ligados à área computacional e que suas estruturas, tanto de história quanto de tecnologia, são fundamentais para seu sucesso.

Com todos esses conceitos apresentados, é notável o poder de aprendizado que o desenvolvimento de jogos pode proporcionar desde conceitos gráficos até o enredo. Mas seu principal destaque é o desenvolvimento de seu núcleo que vai ditar de maneira geral todo o funcionamento de um jogo. O núcleo por sua vez é desenvolvido com linguagens de programação, envolvendo diretamente a necessidade da aplicação de lógica sendo, portanto, a área ideal para que os alunos desenvolvam essa habilidade e, ao mesmo tempo, consigam visualizar o resultado do que desenvolveram dentro do jogo.

2.3 O QUE SÃO *FRAMEWORKS*?

Sobre o que consiste o conceito de *framework*, Johnson e Foote (1988, online) colocam que segundo as suas diversas propostas e diversas definições parciais a respeito do tema trata-se de “um conjunto de classes que incorpora um *design* abstrato para soluções de uma família de problemas relacionados” (Johnson, Foote, 1998, tradução nossa[4]). Esta é a ideia de modularização de soluções para problemas em comum recorrentes na área da computação, voltada para programação orientada a objetos. Outra definição para *framework* é: “um conjunto de objetos que colaboram para executar um conjunto de responsabilidades para um domínio do subsistema de aplicativo. ” (Johnson; Foote, 1988, tradução nossa[5]). Pode-se afirmar que esta definição está mais voltada a ideia de “clusterização” de sistemas. A clusterização pode ser entendida como “o nome dado a um sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa. ” (Alecrim, 2013, online). Partindo deste princípio, a clusterização consiste

---

[4] "A set of classes that incorporates an abstract design for solutions to a family of related problems.”
[5] "A set of objects that collaborate to perform a set of responsibilities for an application subsystem domain."

em componentes de processamento que trabalham em conjunto para a resolução ou construção de um método de trabalho. Este processo de “estrutura” ou “clusterização” é considerado como um único sistema. Uma analogia que pode ser feita com a ideia de *clusters* é a de uma “linha de montagem ou produção” ou as “estruturas de uma edificação”.

Adentrando ainda mais no conceito de frameworks, trata-se de “um conjunto de classes abstratas e a maneira como os objetos nessas classes colaboram”. (Johnson; Foote, 1988, tradução nossa[6]). Ou seja, as classes abstratas possuem o encargo e a característica de não possuírem formalidade e serem implícitas adentro a construção de um sistema orientado a objetos. Elas representam espécies de “rascunhos” que servem como modelos, onde classes que herdam delas devem comportar-se de acordo com tal modelo. Continuando na mesma linha de raciocínio, Gamma et al (1994, p. 26) define um *framework* como: “um conjunto de classes cooperantes que compõem um *design* reutilizável para uma classe específica de *software*” (tradução nossa[7]). A impressão que é transmitida pelo entendimento Gamma et al a respeito do tema é uma incrementabilidade aos conceitos estudados por Johnson e Foote sobre *frameworks*. O termo chave nesta questão é a “cooperação” entre as diferentes classes que se comunicam em um sistema e possuem enlaces para aplicarem a reutilização, já adentrando a camada mais técnica de aplicações, o código e o paradigma da orientação a objetos de desenvolvimento. Já de acordo com Rogers (1997, pg 4), *framework* “é uma biblioteca de classes que captura padrões de interação entre objetos. Um *framework* consiste em um conjunto de classes concretas e abstratas, explicitamente projetadas para serem usadas juntas. ” (Tradução nossa[8]).

Neste contexto, Rogers (1997) admite que a tal definição de estrutura colocada por diversos autores, agora está para uma biblioteca que possui um conjunto de instruções e funções já modularizadas e pré-prontas para serem usadas em uma aplicação de modo externo.

Então, como apresentado, é possível diversas interpretações para *framework*, mas há uma convergência conceitual entre diversos autores da área da computação a respeito do tema, em que um *framework* é uma aglomeração de funcionalidades

---

[6] "A set of abstract classes and the way objects in those classes collaborate."
[7] "A set of cooperating classes that make up a reusable design for a specific class of software."
[8] "Is a class library that captures patterns of interaction between objects. A framework consists of a set of concrete and abstract classes, explicitly designed to be used together."

modulares para a resolução de problemas comuns, mais especificamente da área da Informática.

Logo, outro conceito a se considerar é o conceito de biblioteca, que segundo Zanette (2017) disponibilizam funções prontas o que acaba facilitando e acelerando o desenvolvimento permitindo assim o *Clean Code* que facilita a divisão de tarefas possibilitando uma menor repetição de código e também no menor número de dependências.

Ambos os conceitos, *framework* e biblioteca, contribuem para que a agilidade da programação seja garantida por parte dos desenvolvedores, a fim de buscar redução do tempo de desenvolvimento de aplicações e a não preocupação com os recursos de implementação tecnológicos de baixo nível. Como coloca Maniero (2014), os conceitos de *framework* e biblioteca são complementares, pois um *framework* é normalmente constituído por um conjunto de bibliotecas. Mas é importante ter atenção com o seguinte: uma biblioteca pode ser um *framework*, porém um *framework* não pode ser uma biblioteca, ela está contida em um *framework*. Por sua vez, um *framework* pode conter um conjunto de bibliotecas.

Ou seja, Zanette (2017) classifica que o *framework* tem como objetivo central facilitar o desenvolvimento de tarefas repetitivas como, por exemplo, telas de acesso (login) que, no geral, sempre seguem o mesmo conceito de funcionamento. No entanto, quando o desenvolvedor acaba utilizando um *framework* de maneira incorreta ele acaba por realizar o processo inverso, tornando o desenvolvimento e manutenção ainda mais custosos. Já com relação às bibliotecas, Zanette (2017) afirma que elas disponibilizam funções prontas, o que acaba facilitando e acelerando o desenvolvimento, permitindo assim o *Clean Code.* Isto facilita a divisão de tarefas e possibilita uma menor repetição de código e de número de dependências.

Com os conceitos apresentados, é possível chegar a conclusão que tanto as bibliotecas quanto os *frameworks* possuem um objetivo em comum que é o de facilitar o custo de desenvolvimento para o desenvolvedor.

Ainda com relação aos *frameworks*, eles possuem a capacidade de ‘reusabilidade’, ou seja, deve ser do tipo reusável. Porém, ele deve ir além, deve ter também a característica de ‘usabilidade’. Segundo Jacques (2006, online), esta característica tem a ver com um código bem documentado e com facilidade ou praticidade em seu uso. Ainda segundo Jacques, um *framework* deve ter ‘extensibilidade’, ou seja, deve conter uma estratégia perspicaz em sua utilização, o

que significa ter suas classes de modo ‘abstrato’. Além disso, deve haver ‘segurança’, ‘eficiência’ e ‘completude’. A segurança, segundo Jacques (2006, online), se deve ao fato do desenvolvedor não destruir seu *framework* de uso. Já a eficiência, para Jacques (2006, online), é necessária por causa do “seu uso em muitas situações, algumas das quais poderão necessitar de eficiência”. E, por fim, a completude, que transmite a ideia de se endereçar o domínio do problema como um invólucro (Jacques, 2006, Online).

Como exemplo contextual de apresentação, podemos citar um dos mais famosos *frameworks* de desenvolvimento WEB da atualidade: ‘Laravel’. O Laravel (FIGURA 1) é um *framework open-source* de desenvolvimento em linguagem PHP, utilizando o padrão MVC (*Model-View-Controller*). Dentre suas diversas características, o que ele realmente tem como objetivo é o auxílio ao desenvolvimento de aplicações de forma segura e performática, em que o intuito por meio deste é o desenvolvimento de um código mais limpo e padronizado, seguindo boas práticas da programação, com isto os resultados são mais produtivos. A aplicabilidade do Laravel em desenvolvimento web auxilia programadores na construção de sistemas mais robustos, tendo maior praticidade em seu uso.

FIGURA 1 - LOGOTIPO DA TECNOLOGIA PHP (LARAVEL)

FONTE: Zimmermann (2018).

Então, a proposta deste projeto é a construção de uma biblioteca como *framework*, criando recursos próprios e com recursos de biblioteca externos (outros serviços). O objetivo de aplicação deste *framework* é a integração com o ensino, a fim de que esta ferramenta possa auxiliar estudantes iniciantes em programação de computadores.

2.3.1 Framework e sua Classificação

Os *frameworks* estão cada vez mais comuns na área de desenvolvimento, tendo como objetivo central facilitar e acelerar o desenvolvimento, contando com inúmeras variações de uso.

Para Silva (2000 apud ANDRADE, 2009), um *framework* possui diferentes maneiras de ser classificado, variando de acordo com o seu objetivo. Já para Taligent (1994 apud ANDRADE, 2009), entre essas possíveis variações, uma das classificações mais utilizadas e realizada é através do propósito ou escopo, conforme descrito abaixo:

- Aplicação ou de Infraestrutura: é responsável pela funcionalidade de diversos domínios como, por exemplo, em *frameworks* com enfoque em sistemas operacionais.
- Domínio: realiza a cobertura de funcionalidades de um domínio específico, ou seja, tem a preocupação de solucionar problemas específicos. Um possível exemplo seriam aplicações de engenharia financeira.
- Suporte ou de integração de *Midleware*: disponibiliza serviços de baixo nível, sendo utilizado principalmente na comunicação entre sistemas e também na integração de aplicações distribuídas, como por exemplo *frameworks* como o DCOM[9].

Yassin (2000 apud ANDRADE, 2009) já coloca que um *framework* também pode ser classificado de acordo com o processo de reuso de suas classes, podendo ser do tipo:

- Caixa Branca: possui forte ligação com características da orientação a objetos, o seu reuso se decorre através de implementação de herança e métodos.
- Caixa Preta: permite a extensão através do uso de interfaces em componentes que podem ser conectados ao *framework*, tendo o seu

---

[9] DCOM é uma tecnologia proprietária da Microsoft para permitir a comunicação entre componentes de software distribuídos em uma rede.

reuso através da aplicação de composição ou de interfaces para os componentes.

- Caixa Cinza: É a combinação entre os métodos de caixa preta e caixa branca e tem como objetivo superar as desvantagens das demais classificações que permitem pouca flexibilidade no desenvolvimento, possui uma grande flexibilidade e extensibilidade, deixando oculto informações para usuários externos da aplicação.

Com isso, é possível constatar que os *frameworks* possuem diversas classificações que se adaptam às diferentes necessidades de uso visando sempre facilitar e acelerar o desenvolvimento

O *framework* (B.E.A.G.L.E.) por sua vez se encaixa na classificação de caixa cinza, pois além de possuir as características de flexibilidade e extensibilidade de suas funcionalidades possui também o uso de conceitos da orientação a objetos da classificação caixa branca e utiliza interfaces características da classificação caixa preta.

2.3.2 Quem são os atores de um framework?

Nenhum sistema existe isoladamente, todo sistema precisa interagir com um ator humano ou com algum ator automatizado (por exemplo, um outro sistema), que utilizam do sistema para algum propósito e esses atores esperam uma resposta previsível, conforme coloca Booch (1998).

Um *framework* também precisará de um ator para interagir, como mostra o caso de uso do B.E.A.G.L.E. (FIGURA 5). Para tal, na prática, apenas o sistema desenvolvido pelo usuário irá interagir com o *framework*, visto que apenas ele irá se relacionar com as funcionalidades do sistema, neste caso os casos de uso.

2.4 METODOLOGIA DE DESENVOLVIMENTO DE FRAMEWORKS

Visando a possibilidade de reuso e com o foco no aprendizado, o *framework* B.E.A.G.L.E. aplica o paradigma de *framework* orientado a objetos, que para ARANGO (1991 apud SILVA, 2000), visa gerar a descrição de domínios como uma estrutura de classes, o que possibilita o seu reuso em diferentes aplicações e nesse segmento um dos principais pilares é a abordagem de análise de domínio.

A análise de domínio, segundo ARANGO (1991 apud SILVA, 2000), tem a responsabilidade de identificar conjuntos de aplicações que possuem características em comum, gerando assim um domínio de aplicações. A importância da geração de um domínio de aplicações se deve ao objetivo de se reaproveitar o que já foi desenvolvido para que assim se acelere o desenvolvimento da aplicação.

Segundo NEIGHBORS (1991, apud SILVA, 2000), a análise de domínio tem o objetivo de realizar a identificação dos objetos, operações e relações que podem adquirir importância para a descrição do domínio.

Já para ARANGO (1991, apud SILVA, 2000), a análise de domínio é fundamental para a criação de *softwares* reutilizáveis e ressalta, também, que os elementos gerados pela análise de domínio são mais facilmente reutilizados por capturarem a funcionalidade essencial necessária para o funcionamento do domínio.

Para Silva (2000), análise de domínio tem o objetivo de produzir contribuições para a reutilização do *software*. Essas contribuições, por sua vez, equivalem a descrição do domínio contida em um modelo de domínio. Segundo ARANGO (1991, apud SILVA, 2000), as modelagens de domínio podem se basear em modelagens de outras metodologias de desenvolvimento com ou sem adaptações. A seguir são apresentadas três metodologias com base nos autores: HENNINGER (1995), GOMMA (1994) e NEIGHBORS (1991) (apud SILVA, 2000).

- Metodologia de Henninger: realiza o uso de hipertexto, o que possibilita a consulta a projetos previamente desenvolvidos para se aplicar em um projeto em desenvolvimento, e com cada novo projeto desenvolvido o repositório de informações se torna ainda mais consistente, o que facilita ainda mais o desenvolvimento da modelagem de domínio e, também, acelerando o processo de desenvolvimento.
- Metodologia de Gomma: tem o enfoque no desenvolvimento de um ambiente de produção para que assim seja realizada a geração de modelos de domínio. Os modelos de domínio, por usa vez, são repositórios de objetos que podem ser reutilizados e adquiridos no decorrer do desenvolvimento, possibilitando assim uma base de conhecimento mais sólida.
- Paradigma de Draco: um domínio consiste em um conjunto de objetos, operações e regras que estabelecem as conexões possíveis entre esses elementos e cada domínio possuindo uma linguagem específica,

cuja sintaxe e semântica seguem a conformidade com as regras de combinação de elementos (Linguagem Draco).

Com a apresentação das metodologias, ao analisá-las é possível constatar que todas convergem para o conceito de reutilização de código, que acaba gerando uma “base de códigos” que pode vir a ser utilizada em futuras aplicações, gerando, consequentemente, a aceleração do desenvolvimento.

Ao se desenvolver um *framework* é fundamental uma estrutura flexível que permita a fácil modificação e manutenção. A fácil modificação é a facilidade em se realizar alterações no *framework*, de acordo com as necessidades de uma aplicação específica. Já a manutenibilidade, se refere a facilidade em realizar a manutenção no *framework*, pois o mesmo possui uma estrutura de classes de grande complexidade e, portanto, é de fundamental importância o conhecimento da estrutura do mesmo.

Para o desenvolvimento de um *framework* também é fundamental a construção de uma estrutura de classes robusta para que seja possível o desenvolvimento de novas funcionalidades, podendo superar, até mesmo, o domínio planejado, além de permitir a possibilidade de adaptação a um conjunto de diferentes aplicações.

Contudo, segundo JOHNSON (1991, apud SILVA,2000), para que um *framework* possua as propriedades de generalidade, alterabilidade e extensibilidade é fundamental o bom planejamento da arquitetura, sendo esse o conceito de *framework* orientado a objetos, que aplica os seguintes conceitos: (1) de herança para a reutilização de interfaces; (2) de reaproveitamento de código através do uso de classes; (3) de polimorfismo, na definição de classes e métodos, para que se torne possível o uso desse conceito, que pode ser herdado em futuras aplicações através da generalidade de domínio em classes abstratas .

Segundo FAYAD E SCHMIDT (2006), *frameworks* orientados a objetos aumentam a qualidade do *software* desenvolvido, além de permitir a reusabilidade e a disponibilidade de aplicações semiconstruídas. Entre as principais vantagens de *frameworks* orientados a objetos estão:

- Modularidade: realiza o armazenamento do processamento dentro de relações estáveis.

Reusabilidade: permite o reuso de recursos já desenvolvidos em novas aplicações.

- Extensão: fornece métodos que devem ser implementados em cada aplicação específica.
- Inversão do fluxo de Controle: quem realiza os controles de chamada de métodos é o *framework*.

No entanto, apesar das facilidades citadas anteriormente, os *frameworks* orientados a objetos acabam gerando também efeitos colaterais que aninhados com o mal uso podem se agravar ainda mais. Entre os pontos negativos estão:

- Maior tempo de desenvolvimento: o tempo de desenvolvimento se torna maior por conta da abstração necessária da orientação a objetos.
- Integração: a preocupação com a integração entre os componentes deve se tornar algo contínuo.
- Custo de aprendizagem: por conta dos conceitos de Orientação a Objetos, a curva de aprendizagem de um *framework* orientado a objetos é consideravelmente maior.
- Manutenibilidade: o *framework* deve sempre estar pronto para se adequar a possíveis mudanças na aplicação.
- Validação e remoção de defeitos: por possuir uma estrutura mais complexa a procura e remoção de erros se torna mais trabalhosa.
- Eficiência: por aplicar conceitos de Orientação a Objetos uma porcentagem de eficiência da aplicação é perdida.

Com a apresentação desses conceitos é possível chegar a conclusão que os *frameworks* orientados a objetos, além de uma excelente opção para acelerar o desenvolvimento de uma aplicação, por possuir a abstração da orientação a objetos, também é muito eficiente no ensino, pois torna o aprendizado menos custoso por conta da sua semelhança a conceitos presentes no mundo real, sendo esse um dos motivos do *framework* B.E.A.G.L.E. ser desenvolvido nessa estrutura.

Contudo, visando uma estrutura de *framework* que possua o conceito de reutilização de funcionalidades e que, consequentemente, gere um desenvolvimento menos custoso em relação a tempo de desenvolvimento, foi selecionado a Metodologia de Henninger para o desenvolvimento do *framework* B.E.A.G.L.E.

2.5 TRABALHOS RELACIONADOS

A fim de embasar o desenvolvimento do *framework* B.E.A.G.L.E. tomou-se como base duas ferramentas e um trabalho acadêmico, sendo eles, respectivamente, o UNITY, o CONSTRUCT 2 e o trabalho “Um *framework* para criação de jogos voltados para o ensino de lógica de programação” de MEDEIROS (2014). Informações foram levantadas, a fim de verificar as funcionalidades existentes para atender um *framework* com propósito educacional.

A UNITY é um motor gráfico 3D, que junto a sua IDE permite que o usuário realize o desenvolvimento de jogos de forma totalmente gratuita com opções de licenças pagas. Segundo Gasparotto (2014), a ferramenta possui uma grande gama de funcionalidades, o que a torna extremamente poderosa para o desenvolvimento da lógica, uma vez que disponibiliza ferramentas de aprendizado ao desenvolvedor por meio da prática.

O CONSTRUCT 2 é um *framework* para criação de jogos, voltada para pessoas sem domínio de linguagens de programação e iniciantes da área. A ferramenta permite a criação de comandos e eventos lógicos usando um sistema de arrastar e soltar blocos lógicos, isso, para BARASSUOL (2017), torna o CONSTRUCT um aliado para o ensino da programação. Além do fato de trabalhar com termos usados pelas linguagens de programação tradicionais, porém, com uma abordagem mais simples e didática, permitindo uma visão concreta do que se está desenvolvendo.

Por fim, o trabalho acadêmico de MEDEIROS (2014), o *GameFramework,* foi pensado para integrar ferramentas já existentes, afim de criar uma plataforma com o objetivo de ensinar fundamentos de programação por meio de um sistema de arrastar e soltar, resolvendo assim problemas preestabelecidos. O *GameFramework* permite que um programador mais experiente (educador) use funções para criar outros problemas além dos apresentados por padrão.

2.5.1 Análise de Trabalhos Relacionados

Uma análise de funcionalidades e estrutura dos trabalhos relacionados ao B.E.A.G.L.E. foi realizada e levantou-se características desses trabalhos, conforme o Quadro 1.

QUADRO 1 – COMPARATIVO ENTRE FERRAMENTAS RELACIONADAS

Tópico analisado	B.E.A.G.L.E.	Construct 2	GameFramework	UNITY
Disponível em português	Sim	Não	Sim	Sim
Limitado a problemas pré-definidos	Não	Não	Sim	Não
Gratuito	Sim	Sim	Sim	Sim
Framework de Código Aberto	Sim	Não	Sim	Não

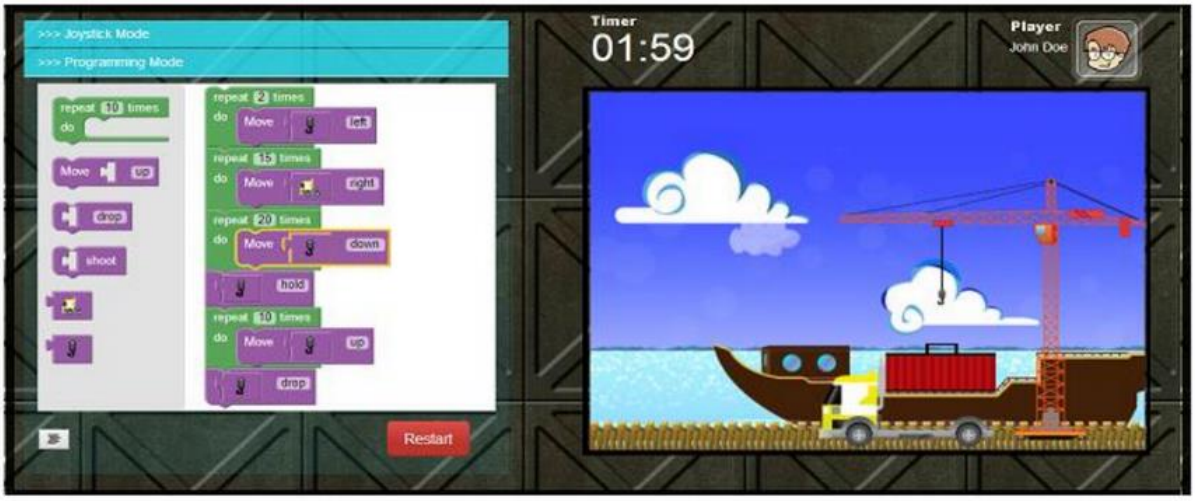
FONTE: Autoria própria (2018).

Dentre as ferramentas analisadas a única não disponível na linguagem português é o CONSTRUCT 2 (FIGURA 2). O *GameFramework* (FIGURA 3) é o único que limita seu usuário a um problema já definido pelo educador, sem dar liberdade de desenvolvimento para o usuário. Todos os *frameworks* são gratuitos apesar da existência de licença paga para o CONSTRUCT 2 e UNITY (FIGURA 4). Por fim, o *GameFramework* e o B.E.A.G.L.E. são *frameworks* de código aberto.

FIGURA 2 – INTERFACE DO FRAMEWORK CONSTRUCT-2

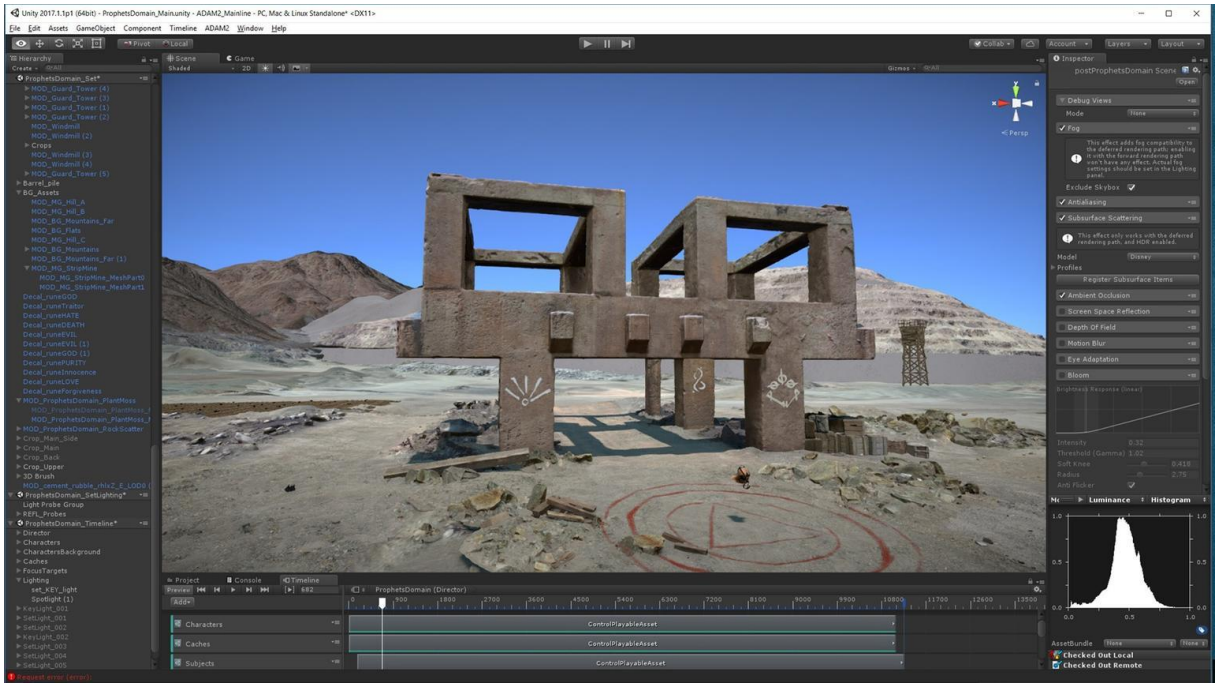
FONTE: Construct-2 (2018)

FIGURA 3 –INTERFACE DO FRAMEWORK GAMEFRAMEWORK



FONTE: Medeiros (2014)

FIGURA 4 –INTERFACE DO FRAMEWORK UNITY



FONTE: Unity (2018).

Os trabalhos analisados possuem objetivos em comum, que é o de proporcionar a seus usuários a possibilidade de desenvolver jogos junto ao aprendizado da lógica.

3 MATERIAIS E MÉTODOS

Nesta seção será apresentado os requisitos do projeto, juntamente com as metodologias aplicadas no escopo do trabalho.

3.1 LEVANTAMENTO DE REQUISITOS

Segundo Campos (2008), o levantamento de requisitos é estágio do desenvolvimento, responsável por indicar e moldar as necessidades do empreendimento. Considerando isso, o levantamento de requisitos para o B.E.A.G.L.E. iniciou na concepção do projeto, a fim de estipular um escopo bem estruturado para a equipe de desenvolvimento seguir.

Também para Campos (2008), o levantamento de requisitos é um método empírico, ou seja, ele não considera o foco nos objetivos do empreendimento. E ainda de acordo com Campos, levando em consideração o paradigma da orientação a objetos, o levantamento de requisitos tem sido feito por meio de Caso de Uso, mas ainda não há técnicas que tornem este método mais sistemático.

Baseado nisso, os requisitos do B.E.A.G.L.E. foram levantados com base em funcionalidades que os desenvolvedores consideraram mais importantes para terem na estrutura de um jogo, além de considerar que se trata de um *framework* voltado para a área da educação. Nessa fase também foi considerado o prazo máximo para o desenvolvimento, filtrando as funcionalidades que iriam constar no *framework*. Tais funcionalidades foram transcritas para os Casos de Uso, bem como as relações entre elas.

Portanto, utilizar do Caso de Uso para levantar os requisitos foi uma alternativa que apresentou bons resultados para o desenvolvimento do *B.E.A.G.L.E.*, levando em consideração que um caso de uso, de acordo com Booch (1998), exemplifica o comportamento do sistema ou de uma parte do sistema e a sua descrição por passos. Como o B.E.A.G.L.E. é separado por funcionalidades distintas cada uma pode ser representada por um caso de uso. Logo, com a aplicação do diagrama de Casos de Uso é possível mostrar como um *framework* funciona de forma transparente e intuitiva.

3.2 ESPECIFICAÇÃO DOS REQUISITOS

A especificação de requisitos é uma parte fundamental no desenvolvimento de um projeto, seja no início, no decorrer ou até mesmo no fim. Segundo Castro (1995 apud Turine et al, 1996) a especificação de requisitos serve para definir uma referência para validar se o que foi feito na etapa de projeto e desenvolvimento está correto de acordo com o escopo do sistema.

No decorrer desta seção são apresentadas as descrições referentes aos requisitos levantados para o sistema. Os Requisitos Funcionais (Quadro 2) e os Requisitos Não Funcionais (Quadro 3).

QUADRO 2 – DESCRIÇÃO DE REQUISITOS FUNCIONAIS

Código do requisito	Nome do requisito	Descrição do requisito
RF001	Criar objeto	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de criar um novo objeto.
RF002	Criar objeto ilustrado	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de criar um novo objeto com imagem definida pelo usuário.
RF003	Criar objeto animado	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de criar um novo objeto com uma animação definida pelo usuário.
RF004	Definir código em etapa de desenvolvimento	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de definir se o projeto está ou não em fase de desenvolvimento para exibir a caixa de colisão de cada objeto da lista.
RF005	Verificar colisão	O sistema que utiliza o *framework* B.E.A.G.L.E. pode verificar se dois objetos estão colidindo.

Código do requisito	Nome do requisito	Descrição do requisito
RF006	Exibir objeto	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a imprimir um objeto criado, sendo animado ou não.
RF007	Verificar restituição entre objetos	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a decidir se dois objetos, colidindo entre si, podem ou não ocupar o mesmo espaço.
RF008	Movimentar objeto	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a movimentar objetos livremente na tela desenvolvida
RF009	Seguir mouse	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a movimentar objetos na direção onde foi clicado em tela.
RF010	Imprimir colisão	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a visualizar a caixa de colisão gerada.
RF011	Controlar rotação de objeto	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a rotacionar um objeto em seu próprio eixo,
RF012	Rotacionar em direção ao mouse	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a rotacionar um objeto direcionando na posição atual do mouse.
RF013	Adicionar imagem	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser apto a adicionar uma imagem a um objeto ilustrado.

Código do requisito	Nome do requisito	Descrição do requisito
RF014	Adicionar animação	O sistema que utiliza o framework B.E.A.G.L.E. deve ser apto a adicionar uma sequência de imagem a um objeto animado.
RF015	Adicionar áudio	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de adicionar um novo áudio.
RF016	Reproduzir áudio	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de reproduzir um áudio adicionado.
RF017	Reproduzir áudio contínuo	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de reproduzir um áudio adicionado, em *loop*.
RF018	Parar áudio contínuo	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de parar a reprodução em *loop* de um áudio adicionado.
RF019	Remover objeto adicionado	O sistema que utiliza o *framework* B.E.A.G.L.E. deve ser capaz de remover um objeto já adicionado.

FONTE: Autoria própria (2018).

QUADRO 3 – DESCRIÇÃO DE REQUISITOS NÃO FUNCIONAIS

Código do requisito	Nome do requisito	Descrição do requisito
RNF001	Utilizar Processing com visão de Java	Utilizar a linguagem de programação Processing, com o modulo de programação Java para desenvolver utilizando a ferramenta B.E.A.G.L.E.

Código do requisito	Nome do requisito	Descrição do requisito
RNF002	Usar a biblioteca de som.	Utilizar a biblioteca de som *Sound* para a execução de áudio.
RNF003	Salvar objeto em lista	Salva o objeto criado (podendo ser normal, ilustrado ou animado) em uma lista de objetos.
RNF004	Simples legibilidade	As funcionalidades desenvolvidas devem ser de fácil entendimento e de uso mais simplificado possível.
RNF005	Traduções	As funcionalidades devem estar em português para melhor entendimento.
RNF006	Caixa de colisão	Todo objeto deve ter uma área para verificar a colisão com a área de outros objetos. Essa área deve ser a multiplicação entre a largura e altura do objeto e deve estar disposta sobre o mesmo.
RNF007	Herança	O objeto animado e o objeto ilustrado devem ser uma especialização de um objeto.
RNF008	Arquivos	Os arquivos de imagem e áudio devem estar em uma pasta abaixo da pasta onde se encontram os programas, intitulada “data”.
RNF009	Arquivos existentes	Todos os nomes de arquivos adicionados devem existir.
RNF010	Reprodução de áudio	Os arquivos de áudio só podem ser reproduzidos após serem criados.

FONTE: Autoria própria (2018).

3.3 METODOLOGIA DE DESENVOLVIMENTO DO *FRAMEWORK B.E.A.G.L.E.*

Ao desenvolver um programa o primeiro passo é escolher uma metodologia de desenvolvimento para se aplicar ao projeto. De acordo com Sommerville (2003 apud SOARES, 2004), metodologias de desenvolvimento são um aglomerado de práticas e resultados que auxiliam no desenvolvimento de um *software*. Dentro da metodologia pode-se definir, por exemplo, os códigos do programa e a análise de requisitos. Como mencionado na seção 2.4, a metodologia de desenvolvimento de Henninger foi aplicada no projeto B.E.A.G.L.E., pois a mesma se encaixa na estrutura do *framework* do proposto.

Segundo HENNINGER (1995 apud SILVA, 2000), a sua metodologia tem como objetivo propor o uso de hipertexto para o armazenamento de todo conhecimento gerado durante o desenvolvimento, que pode a vir ser utilizado em desenvolvimentos futuros. Esse conceito, por sua vez, foi aplicado diretamente no *framework* B.E.A.G.L.E., em que para cada nova funcionalidade desenvolvida, como por exemplo, física ou balística, é realizado o armazenamento da mesma em um repositório junto as demais funcionalidades, que podem vir a ser utilizadas de acordo com as necessidades no decorrer do desenvolvimento do projeto.

Nesse ponto se aplica mais um conceito da metodologia Henninger, que com o uso do hipertexto, permite a consulta as funcionalidades já desenvolvidas de maneira simples, sendo esse o conceito de reutilização, fator também aplicado no *framework* B.E.A.G.L.E., o qual possui o armazenamento de todas as suas funcionalidades e quando é necessário o uso de uma delas, a chamada efetuada de maneira mais simples.

Essas funcionalidades por sua vez foram divididas nos seguintes grupos:

- Funcionalidades estruturais: Criar objeto, criar objeto ilustrado, criar objeto animado e remover objeto

- Funcionalidades gráficas: Definir em desenvolvimento, imprimir objeto, imprimir objeto animado, imprimir colisão, adicionar imagem e adicionar animação

- Funcionalidades para física: Verificar colisão, verificar restituição, seguir mouse, rotacionar ao Mouse, girar objeto e movimentar objeto.

- Funcionalidades de som: Adicionar áudio, tocar áudio, tocar áudio contínuo e parar áudio

Baseado nesse panorama, é possível chegar à conclusão que o uso da metodologia de Henninger permite que o processo de desenvolvimento do *framework* se torne muito mais produtivo devido ao uso de hipertexto, que permite que a reutilização de funcionalidades no projeto se torne muito mais simples, o que, consequentemente, facilita o incremento de novas funcionalidades em versões futuras sendo essa uma das principais características necessárias em um *framework*.

Cada elemento do modelo de domínio proposto por Henninger será apresentado a seguir no Quadro 4.

QUADRO 4 – MODELO DE DOMÍNIO DE HENNINGER

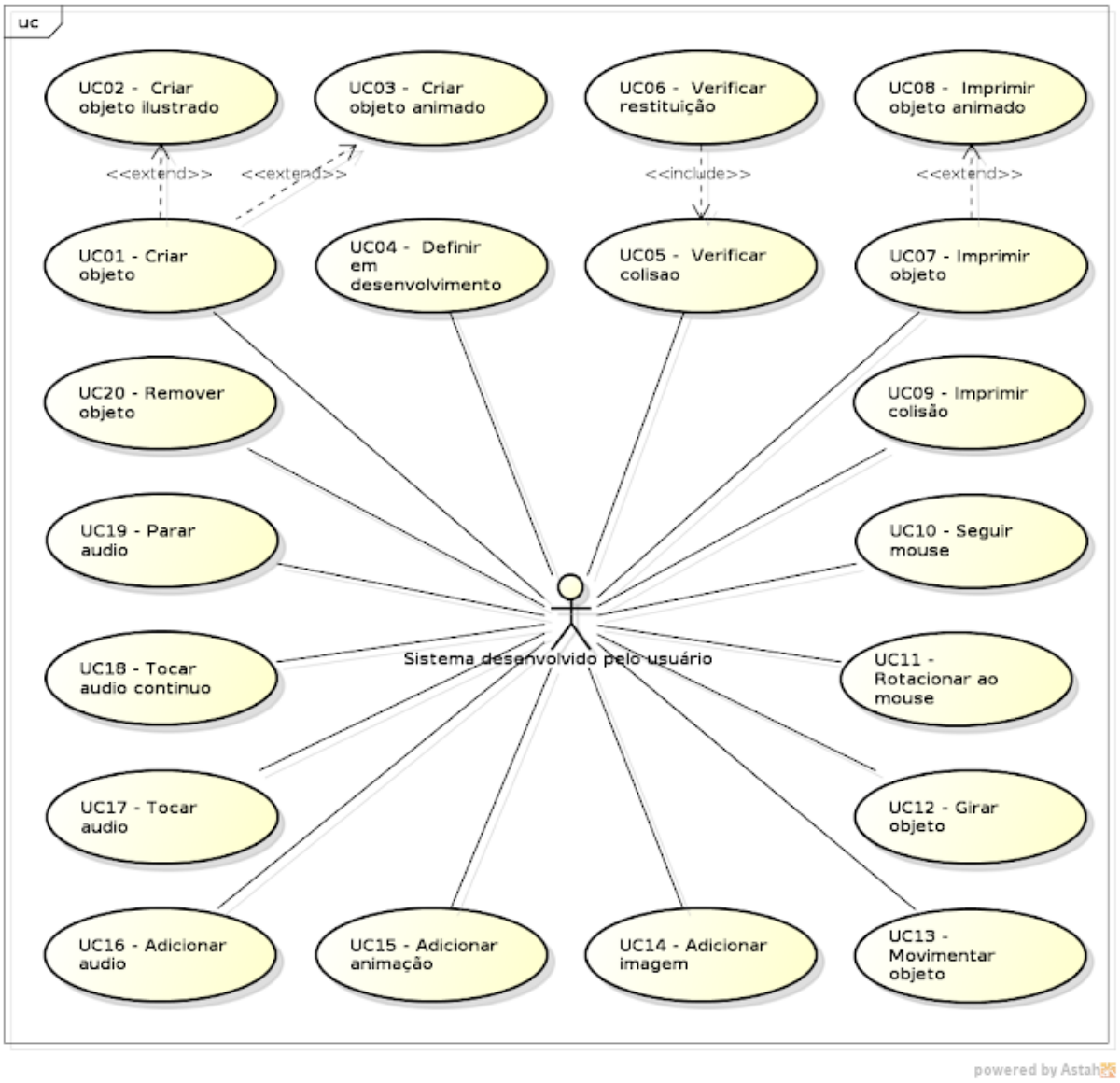
Modelo de Domínio	Objetivo
Repositório de Projetos	É formado por um conjunto de projetos de aplicações já previamente desenvolvidas que podem a vir ser utilizados em desenvolvimentos futuros.
Hipertexto	Tem o objetivo de realizar a sumarização e descrição das funcionalidades da ferramenta. Nesta etapa também devem ser descritas as aplicações do *framework*. Em outras palavras, neste momento devem ser apresentadas as funcionalidades do sistema e o contexto de seu uso.

FONTE: Autoria própria (2018).

Então, para a realização do elemento de hipertexto foi utilizado a diagramação UML para caso de uso que, de acordo com GUEDES (2014), procura identificar e representar os atores que utilizarão o *software* e suas possíveis ações. O diagrama de caso de uso referente ao *framework* B.E.A.G.L.E está ilustrado na FIGURA 5.

Para abordar a necessidade de descrição de funcionalidades que a metodologia eleita demanda, foi aplicado a descrição dos casos de uso, um recurso da modelagem proposta por GUEDES (2014), com o objetivo de descrever como se organizarão métodos previamente diagramados. Com isso facilita o entendimento do desenvolvedor, e cria uma documentação inteligível para o possível usuário deste projeto. As descrições referentes ao diagrama de caso de uso da FIGURA 5 encontram-se no Apêndice 1.

FIGURA 5 – DIAGRAMA DE CASO DE USO



FONTE: Autoria própria (2018).

Os hipertextos foram gerados de acordo com a característica de cada nova funcionalidade desenvolvida facilitando assim a utilização da mesma em partes do projeto em que seu uso é necessário. Já os repositórios foram gerados se levando em consideração o objetivo de cada funcionalidade deixando por exemplo os métodos ligados a gravidade em um mesmo repositório seguindo esse mesmo esquema de armazenamento nas demais funcionalidades.

O segundo ponto abordado por Henninger (1995 apud SILVA, 2000) é o uso de metalinguagem contextualizada sob o desenvolvimento. Em outras palavras,

Henninger defende que deve haver uma biblioteca com projetos de aplicações já desenvolvidas previamente.

Ainda, seguindo Guedes (2014), o diagrama de classes é uma representação do sistema em modo gráfico, que lista o fluxo técnico de operações das funcionalidades modeladas. É o diagrama mais próximo da implementação, onde são apresentados os métodos, parâmetros e atributos de cada classe do sistema. O diagrama de classes converge com a metodologia proposta por Henninger no momento onde temos a necessidade de um embasamento técnico prévio de aplicações. Com o diagrama de classes, possuímos uma visão macro e, ao mesmo tempo, técnica de toda a ferramenta, compilado em um único escopo gráfico. A modelagem das classes do *framework* B.E.A.G.L.E encontra-se no Apêndice 2.

Por fim, a metodologia visada por Henninger (1995 apud SILVA, 2000), necessita de um último ponto para completar o modelo de domínio. Para ele, se torna necessário a existência de uma conjuntura de programas prontos, para que o uso da metalinguagem seja congruente dentro do modelo proposto. Em outras palavras, é necessário um repositório de programas que implementem a ferramenta desenvolvida para exemplificar o bom uso desta, apresentando, aos consumidores do produto criado, um *know-how* de sua utilização.

Para garantir a entrega de tal necessidade dentro da metodologia proposta, foi implementado um jogo, o qual é apresentado no Capítulo 4 e o código encontra-se no Apêndice 3.

Com um embasamento e um repositório de projetos desenvolvidos, atinge-se o último passo da metodologia abordada neste projeto. Por fim, temos a finalização da metalinguagem oferecendo exemplos de uso em um contexto centrado em jogos.

3.4 GERÊNCIA DO PROJETO

Segundo o ALVES (2006 apud Project Management Institute, 2000), um projeto é um esforço temporário com o intuito de criar um bem ou serviço singular. Baseado nisso, a gerência de projetos é o ato de aplicar práticas, experiência, técnicas e ferramentas para que um projeto possa atingir seu objetivo (ou ir além dele).

Sendo assim, o Diagrama de *Gantt* foi a ferramenta escolhida para gerenciar as tarefas do projeto B.E.A.G.L.E. e também para gerenciar como o tempo foi utilizado para a confecção de cada etapa do projeto. Segundo Jones (1988, tradução nossa) o Diagrama de *Gantt* é amplamente usado para representar planos de produção, agendas e desempenho. Nele cada atividade é representada por um retângulo e cada retângulo tem uma largura que é proporcional a unidade que consta no plano em que o diagrama se encontra. A Figura 6 mostra o Diagrama de *Gantt* do projeto B.E.A.G.L.E., com a unidade de tempo em dias.

FIGURA 6 – DIAGRAMA DE GANTT DO B.E.A.G.L.E.

#	Atividade	Início	Duração	Término
1	Desenvolvimer	10/08/2018	5	15/08/2018
2	Desenvolvimer	15/08/2018	4	19/08/2018
3	Desenvolvimer	19/08/2018	4	23/08/2018
4	Desenvolvimer	23/08/2018	13	05/09/2018
5	Desenvolvimer	05/09/2018	5	10/09/2018
6	Desenvolvimer	10/09/2018	8	18/09/2018
7	Desenvolvimer	18/09/2018	6	24/09/2018
8	Desenvolvimer	24/09/2018	9	03/10/2018
9	Desenvolvimer	03/10/2018	8	11/10/2018
10	Desenvolvimer	11/10/2018	5	16/10/2018
11	Desenvolvimer	16/10/2018	3	19/10/2018
12	Desenvolvimer	19/10/2018	3	22/10/2018
13	Desenvolvimer	22/10/2018	15	06/11/2018

FONTE: Autoria própria (2018).

Baseado no Diagrama de Gantt (Figura 6) foi elaborado um quadro que contém o responsável por cada atividade no desenvolvimento do Framework B.E.A.G.L.E. apresentado no Quadro 5 a seguir.

QUADRO 5 – DIVISÃO DE RESPOSABILDIADES NO DESENVOLVIMENTO

Responsável	Atividade
Gabriel Vieira de Queiroz	Desenvolvimento das ferramentas: Verificar Colisão, Verificar Restituição, Seguir Mouse, Rotacionar ao Mouse, Girar Objeto e Movimentar objeto.

Lucas Alves Coutinho Gehlen	Criar Objeto, Criar objeto ilustrado, Criar objeto animado, Definir em desenvolvimento, Imprimir Objeto, Imprimir objeto animado, Imprimir colisão, Adicionar imagem, Adicionar animação, Adicionar áudio, Tocar áudio, Tocar áudio contínuo, Parar Áudio e Remover objeto.

FONTE: Autoria própria (2018).

Baseado no diagrama a equipe desenvolveu o projeto, e conforme o passar do tempo houve a possibilidade de decidir o tempo máximo para cada tarefa, além de prospectiva de incluir ou não mais funcionalidades dentro do framework, assim garantindo que o projeto fosse entregue até a data máxima.

Portanto, ao se utilizar do Diagrama de Gantt pode-se entender o contexto temporal em que as partes do projeto se encontram, podendo assim utilizar do diagrama para a tomada de decisões, além de mostrar se o tempo está sendo usado de forma correta para a desenvoltura das atividades.

3.5 TECNOLOGIAS UTILIZADAS

O *Processing* consiste em uma linguagem de programação para desenvolvimento de arte eletrônica e processos visuais (Processing, Online, 2018), a fim de servirem como recursos visuais que envolvem grafia e abstracionismo gráfico, conceitos estes mais aprofundados associados ao tema artístico. Sua sintaxe é baseada na linguagem C, porém traz os recursos de desenvolvimento de *software* orientado a objetos. Consiste em uma tecnologia *open source*, gratuita e de código aberto, servindo e rodando em diversas plataformas como *Mac*, *Windows* e GNU/Linux, possuindo um ambiente integrado de desenvolvimento (Processing, Online, 2018).

O desenvolvimento de aplicações gráficas (desenho e esboço geométrico, incluindo animações) faz com que o aprendizado de conceitos relacionados aos processos gráficos e visuais sejam aperfeiçoados, a fim de se buscar a integração junto a programação orientado a eventos ou objetos (Processing, Online, 2018).

Inicialmente, o *Processing* foi direcionado para a área de artes visuais, “alfabetização visual dentro da tecnologia” (Processing, Online, 2018). A ideia por trás do *Processing* sempre foi a de um caderno de esboços e criação de desenhos, de

forma representativa digital. Desta maneira seria possível a transmissão dos conhecimentos e fundamentos da programação usando um contexto visual antes de se aprofundar na parte mais teórica de desenvolvimento (Processing, Online, 2018).

Sua IDE permite que o desenvolvedor veja, em um contexto visual, o resultado de seu desenvolvimento. Por esse motivo, é muito recomendada para iniciantes na área da programação. Além disso, é a precursora de muito projetos, entre eles o Arduino. Por isso, a interface deles é muito semelhante, inclusive o próprio *Processing* possui bibliotecas que permitem o desenvolvimento de projetos com o uso dos conceitos de Arduino.

O processamento da linguagem é feito em linguagem Java, com a inclusão de algumas funcionalidades adicionais: classes, funções matemáticas e operações. Antes de ser compilado todo o código é convertido para JAVA puro fazendo com que todas as classes adicionadas sejam tratadas como classes internas.

Já o *hardware* utilizado para o desenvolvimento do *framework* B.E.A.G.L.E. foram cinco *notebooks* e um *desktop*. Suas configurações são descritas a seguir:

a) *Desktop*
   - Sistema operacional: Windows 10™ Pro
   - Memória RAM: 16GB DDR4
   - Armazenamento: 1TB HDD
   - GPU: Nvidia® Geforce® 1060 6GB
   - Processador: Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz
   - Resolução de Tela: 1920x1080

b) *Notebook* Acer VX5-591G
   - Sistema operacional: Windows 10™
   - Memória RAM: 8GB DDR4
   - Armazenamento: 1TB HDD
   - GPU: Nvidia® Geforce® 1050 4GB
   - Processador: Intel® Core™ I5-7300HQ 2.50GHz
   - Resolução de Tela: 1920x1080

c) *Notebook* Acer Aspire E15
   - Sistema operacional: Windows 10™

- Memória RAM: 6GB DDR3
- Armazenamento: 500GB HDD
- GPU: Intel® HD Graphics 4400 1792MB
- Processador: Intel® Core™ I5-4210U 1.7GHz to 2.7GHz (Turbo Boost)
- Resolução: 1366x768

d) *Notebook* Dell Inspiron 5000
   - Sistema operacional: Windows 10™/ Linux Ubuntu™
   - Memória RAM: 8GB DDR3
   - Armazenamento: 1TB HDD
   - GPU: Nvidia® Geforce® GT930M
   - Processador: Intel® Core™ i5-6200U CPU @ 2.30GHz 2.40GHz
   - Resolução: 1366x768

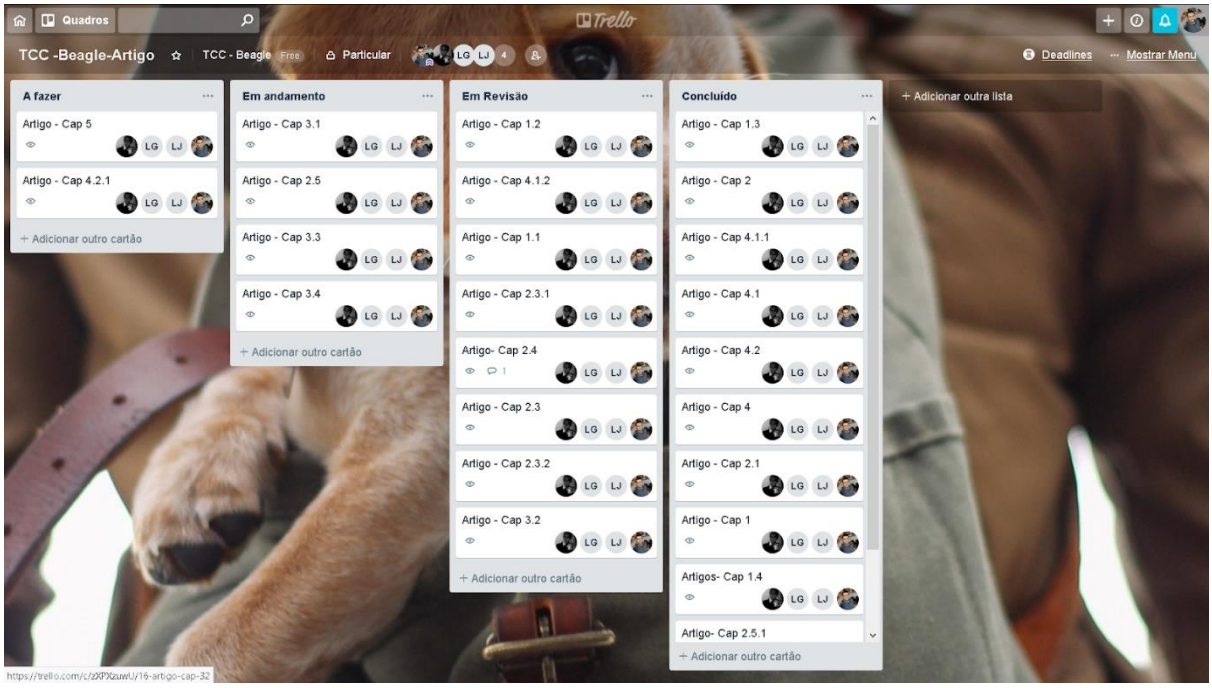
e) *Notebook* Dell Latitude 3457
   - Sistema operacional: Linux Ubuntu™ 16.04
   - Memória RAM: 8GB DDR3
   - Armazenamento: 120GB SSD
   - GPU: Intel® HD Graphics 520 (Skylake GT2)
   - Processador: Intel® Core™ i5-6200U CPU @ 2.30GHz × 4
   - Resolução: 1366x768

Para o desenvolvimento da documentação do projeto foi utilizado o *Google Docs*, que permitiu a produção coletiva de toda a equipe. Além de todos atuarem em simultâneo na construção da documentação do trabalho. A ferramenta também permitiu a fácil interação com os dados armazenados dentro do *Google Drive*, facilitando a inserção dos dados necessários no documento.

Para a divisão e controle das tarefas do projeto foi utilizado duas ferramentas. O *Trello* (FIGURA 7), que além de suas principais funcionalidades serem gratuitas, também permite que as tarefas sejam distribuídas de uma forma clara e objetiva, facilitando assim o desenvolvimento. E, como já mencionado na seção 3.4, o diagrama *Gantt* (FIGURA 6), o qual foi construído utilizado o *software Microsoft Excel* pois, com

o fornecimento dos dados, por período de desenvolvimento de cada funcionalidade, em tabela é possível a geração do diagrama.

FIGURA 7 – PROJETO DO B.E.A.G.L.E. NO TRELLO



FONTE: Autoria própria (2018).

Com relação a modelagem do *framework* B.E.A.G.L.E foi utilizado o *software Astah Community*, o qual foi escolhido por ser uma ferramenta extremamente completa e que também disponibiliza o *software* na íntegra e totalmente gratuito para estudantes.

Por fim, para as atividades de desenvolvimento do *framework* foi utilizada a IDE *Processing*, disponibilizada gratuitamente pela própria desenvolvedora da linguagem. E para o versionamento do código foi utilizado o *GitHub*, aplicativo web gratuito que permitiu que a equipe trabalhasse de forma simultânea no projeto.

4 EXEMPLO DE APLICAÇÃO UTILIZANDO O *FRAMEWORK* B.E.A.G.L.E

Este capítulo tem como objetivo apresentar o jogo implementado utilizando o *framework* B.E.A.G.L.E., a fim de validar a estrutura da ferramenta proposta.

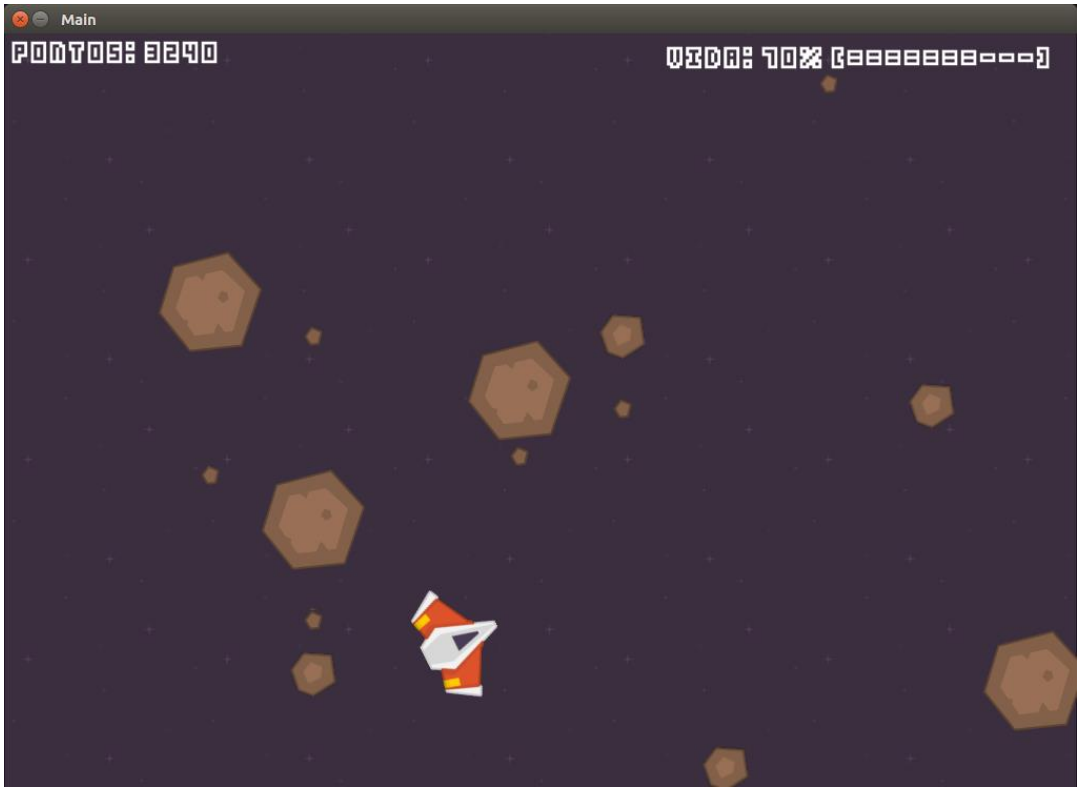
O jogo consiste em uma espaçonave controlada pelo usuário, por meio do mouse, no coração de uma chuva de meteoros, com o objetivo de desviar ou destruir os asteroides que aparecem na tela. Com uma implementação simples, o jogo consegue evidenciar a facilidade que o *framework* B.E.A.G.L.E oferece para resolver problemas complexos, como colisão ou impressão de imagem.

O jogo é intitulado “Chuva de meteoros - Um jogo B.E.A.G.L.E.”, e oferece três telas ao usuário: (1) inicialização, (2) execução do jogo e (3) encerramento. As Figuras 8, 9 e 10 representam, respectivamente, as telas mencionadas.

FIGURA 8 – APLICAÇÃO, TELA DE INICIALIZAÇÃO

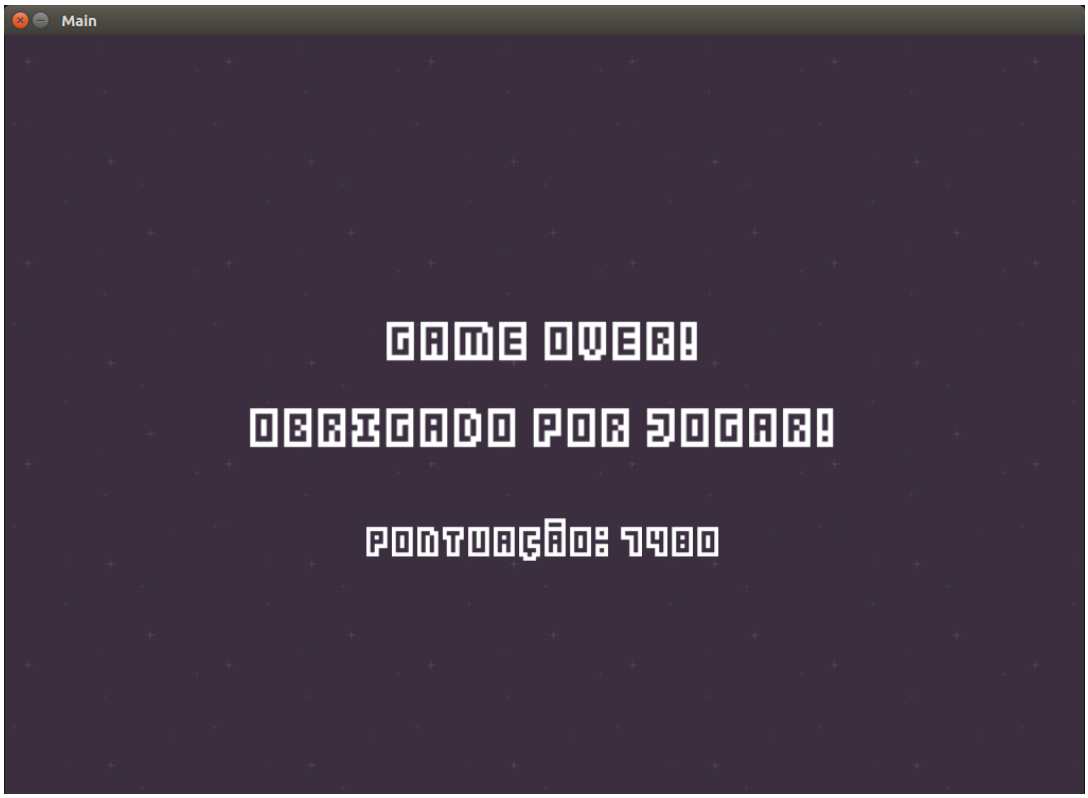
FONTE: Autoria própria (2018).

FIGURA 9 – APLICAÇÃO, TELA DE EXECUÇÃO



FONTE: Autoria própria (2018).

FIGURA 10 – APLICAÇÃO, TELA DE ENCERRAMENTO



FONTE: Autoria própria (2018).

O código gerado, presente no Apêndice 3, contém elementos que aplicam o *framework* B.E.A.G.L.E. desenvolvido. Uma das funcionalidades é a criação de um novo objeto. Por exemplo, para que seja possível criar e exibir a espaçonave é necessário apenas a resolução da chamada de três funcionalidades. A primeira, apresentada na Figura 11, remete a criação da nave como um novo objeto do tipo animado. A segunda chamada, apresentada na Figura 12, é a adição de uma sequência animada para o objeto de espaçonave. Por fim, na Figura 13 temos a última funcionalidade, que apresenta em tela o objeto criado.

FIGURA 11 – CÓDIGO, ADICIONAR ESPAÇONAVE

```
ObjetoAnimado nave  = criarObjeto(OBJETO_ANIMADO, 75, 112, width/2, height/2, 0);
```

FONTE: Autoria própria (2018).

FIGURA 12 – CÓDIGO, ADICIONA ANIMAÇÃO

```
adicionaAnimacao("PLAYERSHIP", "playerShip", 1, nave);
```

FONTE: Autoria própria (2018).

FIGURA 13 – CÓDIGO, IMPRIME ESPAÇONAVE

```
imprime("PLAYERSHIP", nave);
```

FONTE: Autoria própria (2018).

Outra funcionalidade da ferramenta, implementada no jogo “Chuva de meteoros - Um jogo B.E.A.G.L.E.” é a validação de colisão. Tal método necessita de dois objetos criados para realizar a verificação se estes estão colidindo. Sem a necessidade de cálculo de área para cada objeto, a ferramenta disponibiliza um comando simples para a execução da funcionalidade demonstrado na Figura 14.

FIGURA 14 – CÓDIGO, VERIFICA COLISÃO

```
estaColidindo(nave, asteroide);
```

FONTE: Autoria própria (2018).

Uma vez pronto, o jogo visa explanar a utilização das ferramentas implementadas pelo *framework.* Com objetivo único de demonstrar formas de utilizar o projeto desenvolvido, o código do jogo apresenta funcionalidades únicas da ferramenta, enquanto sua interface captura sua real execução. Deste modo, “Chuva de meteoros - Um jogo B.E.A.G.L.E.” serve de embasamento aos desenvolvedores que utilizarão o produto deste projeto, exemplificando suas principais funcionalidades.

4.1 ANÁLISE DE RESULTADOS

Visando a criação de mais repositórios propostos pela metodologia de Heninnger, há mais dois jogos implementados além do “Chuva de meteoros - Um jogo B.E.A.G.L.E.”, cada um trabalhando funcionalidades de forma diferentes a fim de melhorar os repositórios para o *framework*.

Um dos jogos se chama “Canon”, um jogo que pode ser jogado por até dois jogadores e tem como objetivo uma acertar tiros de um tanque de guerra no outro para destruir o tanque de guerra adversário utilizando angulação e força definidos pela posição do ponteiro do mouse.

Já o outro jogo se chama “Snake” e tem como objetivo controlar uma galinha com o teclado para coletar moedas. A cada moeda coletada a galinha ganha um pintinho que seguirá ela a todo instante e quanto mais moedas maior se torna a fila de filhotes. Além de coletar as moedas a galinha não pode encostar em nenhum pintinho nem nas bordas. A seguir neste tópico é demonstrado como funcionam algumas funções dentro dos jogos já abordados e seus respectivos resultados.

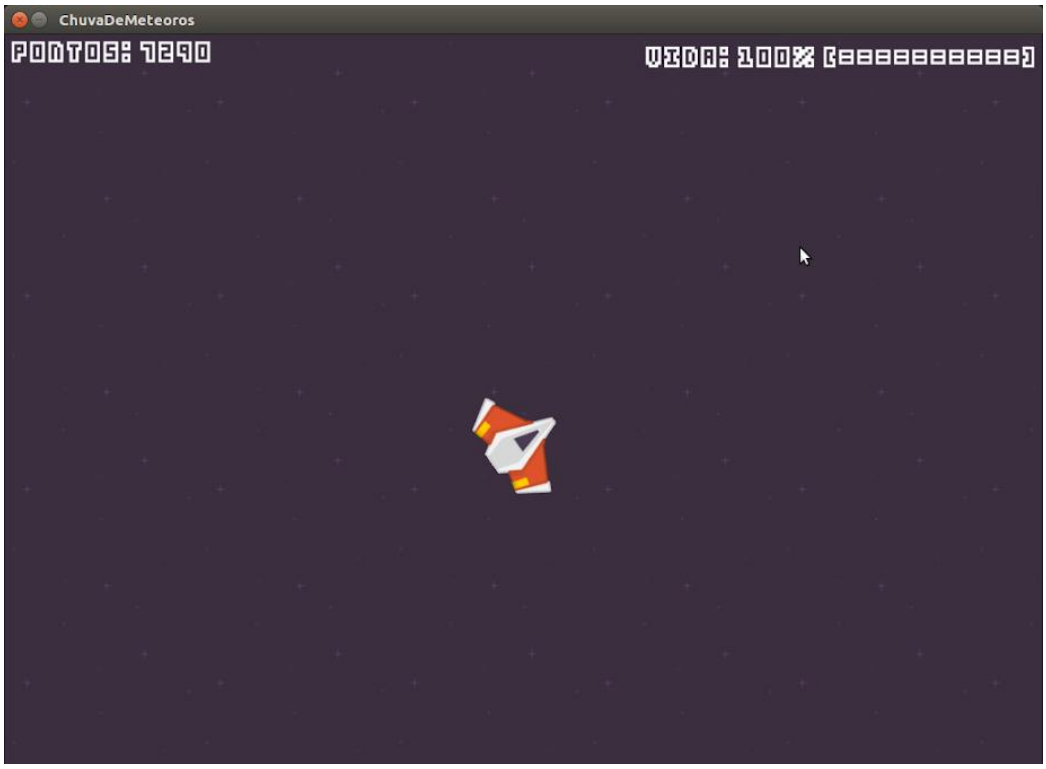
Voltando ao “Chuva de meteoros - Um jogo B.E.A.G.L.E.” o código da Figura 15 remete a função se seguir o mouse representado pelas figuras 16 e 17. Neste trecho é passada a velocidade e qual objeto seguirá o cursor do mouse.

FIGURA 15 – CÓDIGO, FAZER NAVE SEGUIR MOUSE

```
segueMouse(0.05, ship);
```

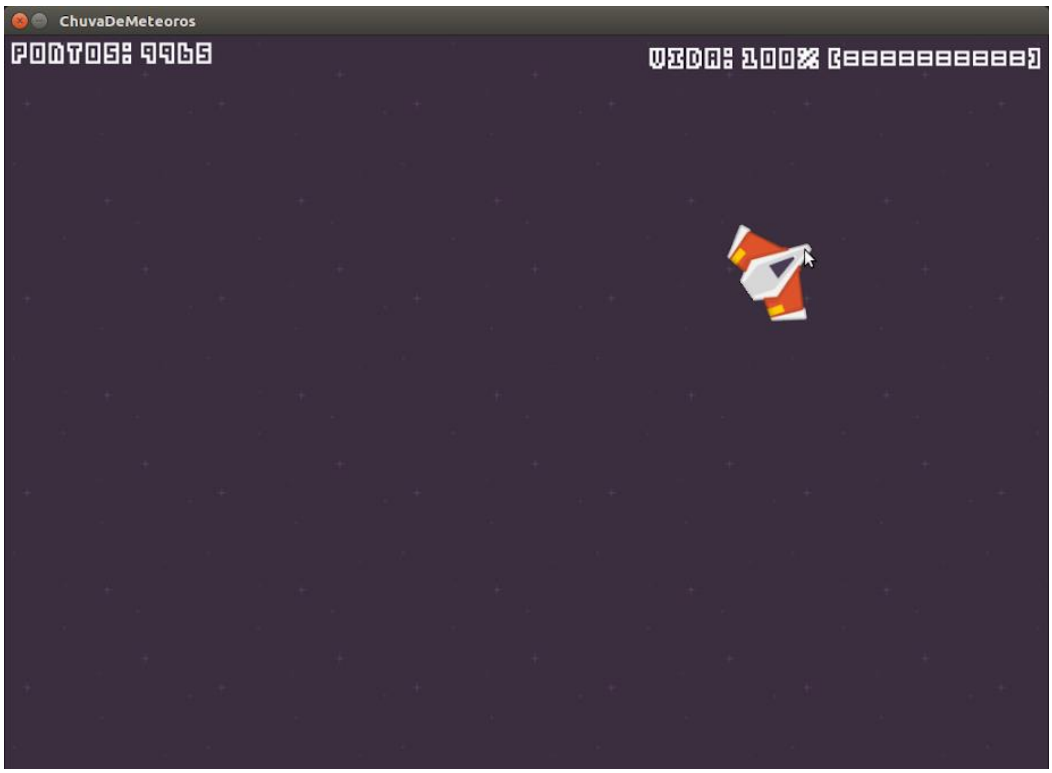
FONTE: Autoria própria (2018).

FIGURA 16 – APLICAÇÃO, POSIÇÃO INICAL DA NAVE



FONTE: Autoria própria (2018).

FIGURA 17 – APLICAÇÃO, POSIÇÃO FINAL DA NAVE



FONTE: Autoria própria (2018).

Indo para outra funcionalidade, porém ainda no mesmo jogo, há o modo desenvolvedor, este que imprime na tela a caixa de colisão de cada objeto, como é mostrado na Figura 19. O trecho de código apresentado na Figura 18 representa a função que recebe um parâmetro de verdadeiro ou falso para ativar ou desativar o modo.

FIGURA 18 – CÓDIGO, ATIVAR MODO DESENVOLVEDOR

```
defineEmDesenvolvimento(true);
```

FONTE: Autoria própria (2018).

FIGURA 19 – APLICAÇÃO, MODO DESENVOLVEDOR ATIVADO

FONTE: Autoria própria (2018).

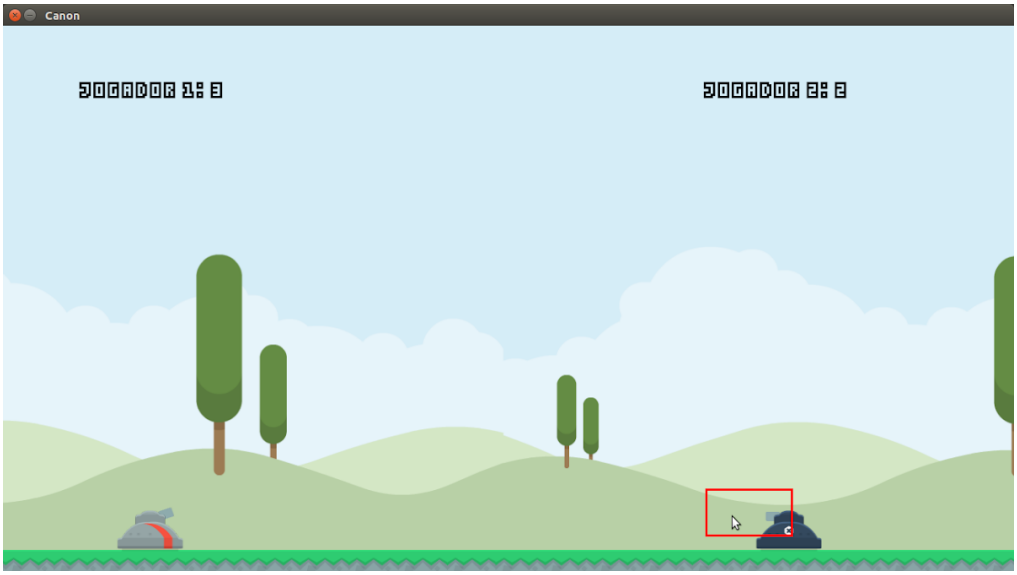
Agora tratando do jogo “Canon” tem-se a função de rotacionar para o mouse, tratada na função representada na Figura 20. Nesta função apenas é passado o objeto que irá rotacionar em direção ao mouse. O resultado pode ser visto nas figuras 21 e 22.

FIGURA 20 – CÓDIGO, ROTACIONAR PARA O MOUSE

```
rotacionarParaMouse(canhao);
```

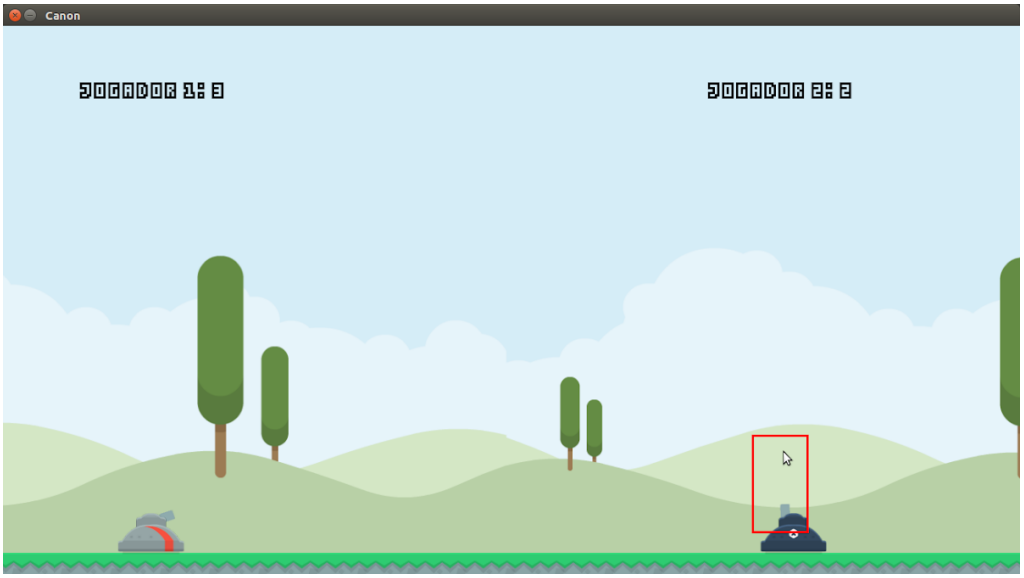
FONTE: Autoria própria (2018).

FIGURA 21 – APLICAÇÃO, POSIÇÃO INICIAL DO CANHÃO



FONTE: Autoria própria (2018).

FIGURA 22 – APLICAÇÃO, POSIÇÃO FINAL DO CANHÃO



FONTE: Autoria própria (2018).

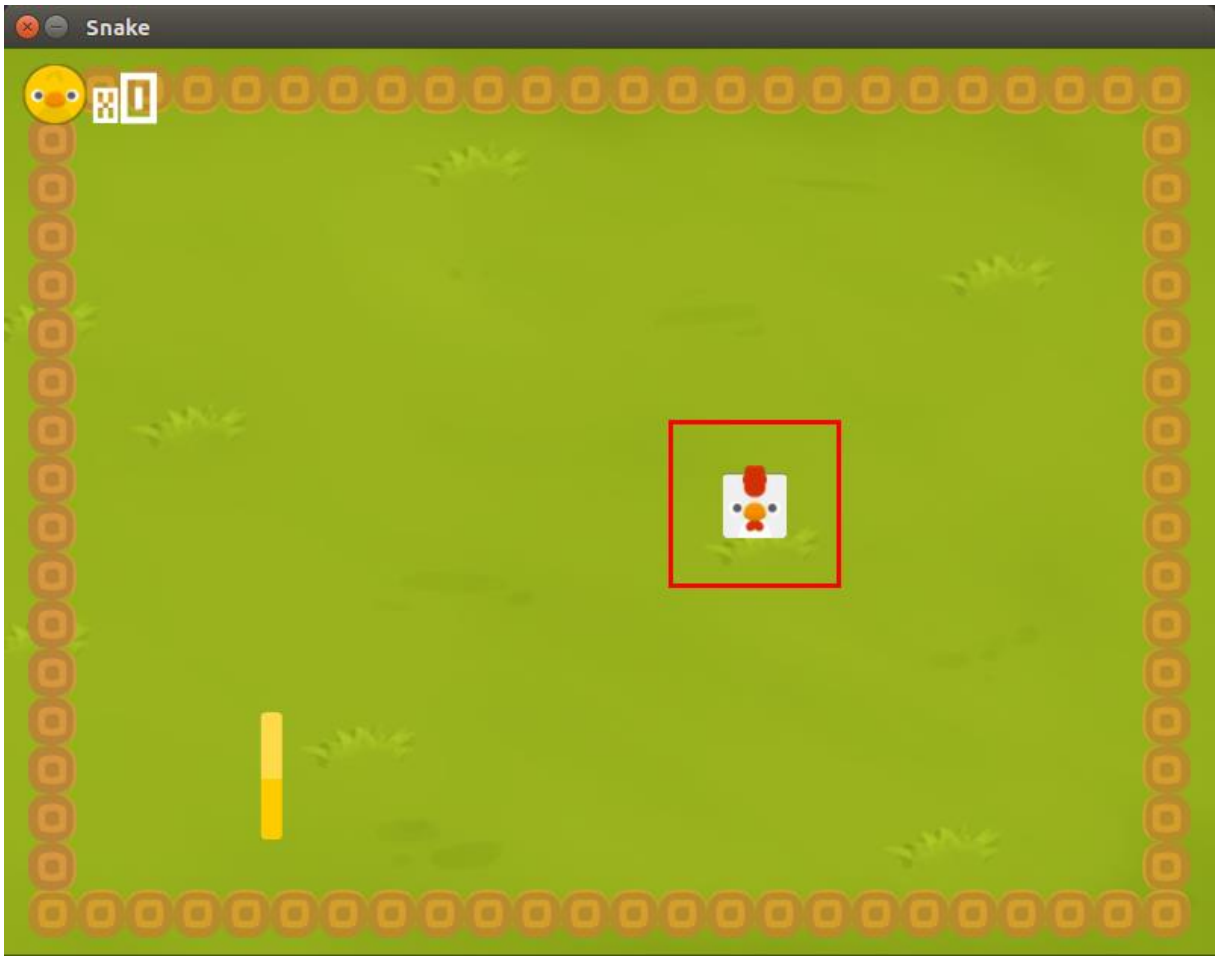
Por fim falando sobre o jogo “Snake” há a função de construção de um objeto desde a criação até a impressão representado no trecho de código da Figura 23. Primeiramente um objeto ilustrado é instanciado e em seu construtor é definido as suas medidas em largura e altura. Então é usada uma função para atribuir uma imagem a este objeto. Por fim é usada a função de imprimir um objeto em tela. O resultado desse trecho é ilustrado na Figura 24.

FIGURA 23 – CÓDIGO, CONSTRUÇÃO E IMPRESSÃO DE OBJETO ILUSTRADO

```
rotacionarParaMouse(canhao); ObjetoIlustrado galinha = criarObjeto(OBJETO_ILUSTRADO, 42, 48);
adicionaImagem("chicken.png", galinha);
imprime(galinha);
```

FONTE: Autoria própria (2018).

FIGURA 24 – APLICAÇÃO, IMPRESSÃO DO OBJETO GALINHA



FONTE: Autoria própria (2018).

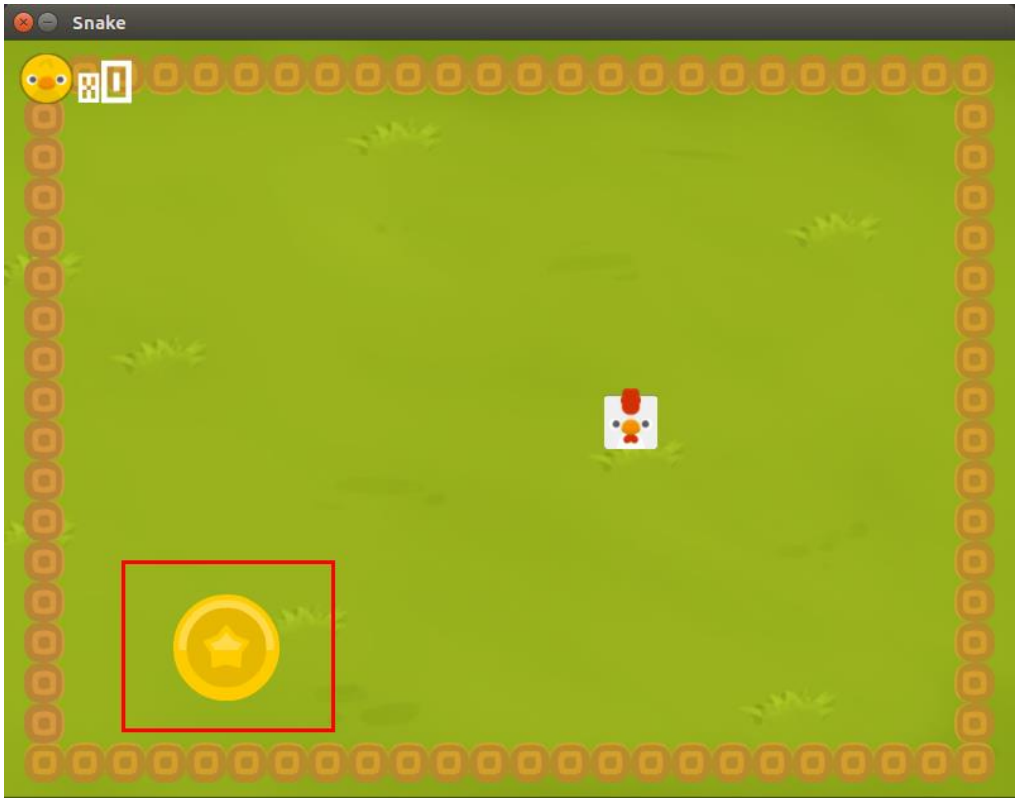
Ademais também há a criação de objeto animado dentro do jogo também. A construção dele se assemelha, como mostra na Figura 25, com a do objeto ilustrado visto anteriormente. Primeiramente é instanciado um objeto animado definindo as suas medidas em largura e altura como parâmetro no construtor. Então é atribuída uma animação para o objeto passando o nome do arquivo, a quantidade de quadros (que estejam dentro da quantidade de arquivos escolhidos) e o objeto a ser vinculado. Por fim é impresso o objeto da mesma forma como foi feito anteriormente. Os resultados são ilustrados nas figuras 26, 27, 28, 29, 30 e 31.

FIGURA 25 – CÓDIGO, CONSTRUÇÃO E IMPRESSÃO DE OBJETO ANIMADO

```
ObjetoAnimado moeda  = criarObjeto(OBJETO_ANIMADO, 84, 84);
adicionaAnimacao("MOEDA", "gold_", 6, moeda);
imprime(moeda);
```

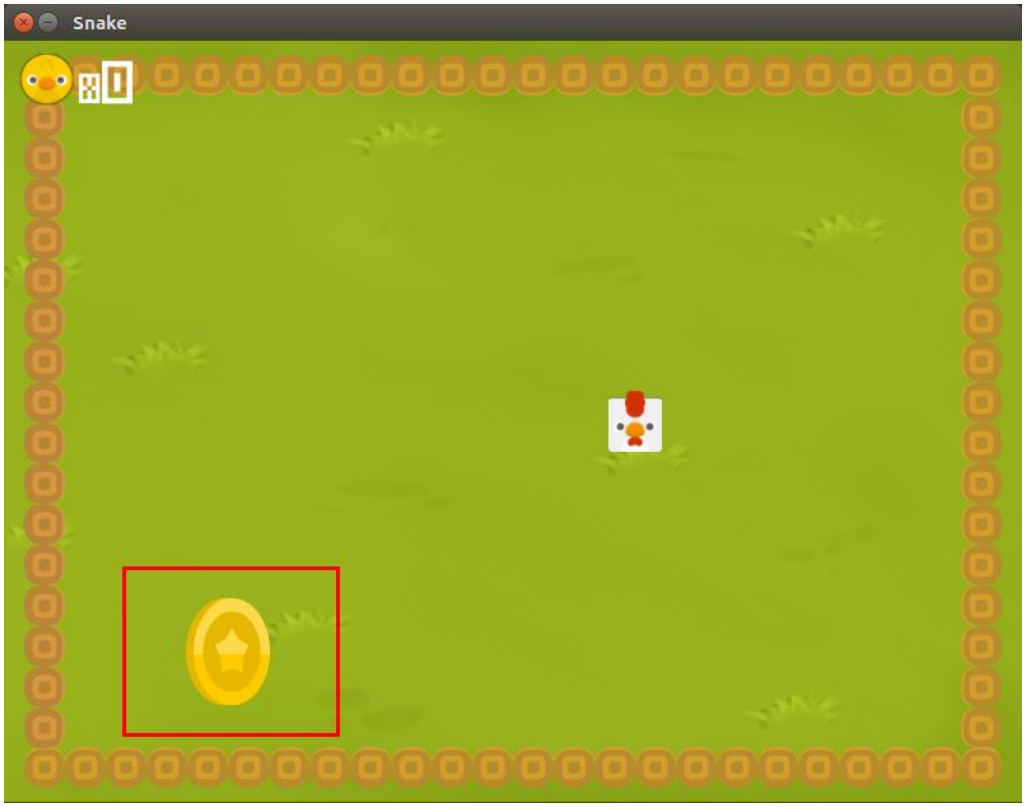
FONTE: Autoria própria (2018).

FIGURA 26 – APLICAÇÃO, PRIMEIRO QUADRO DO OBJETO MOEDA



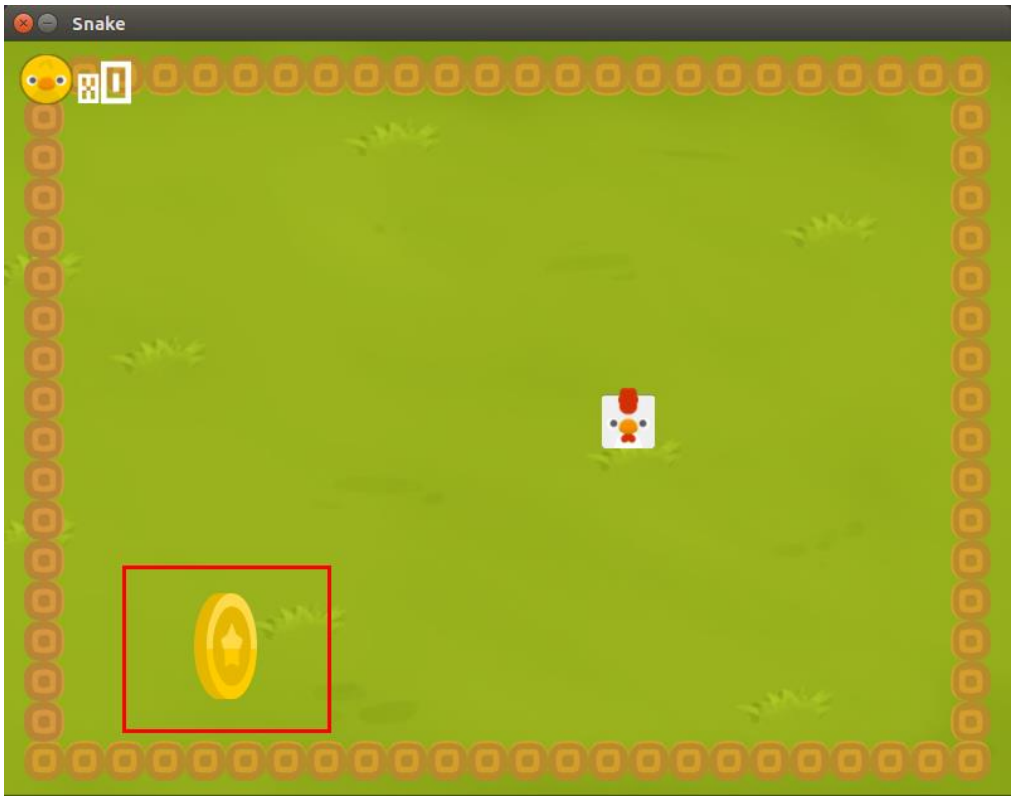
FONTE: Autoria própria (2018).

FIGURA 27 – APLICAÇÃO, SEGUNDO QUADRO DO OBJETO MOEDA



FONTE: Autoria própria (2018).

FIGURA 28 – APLICAÇÃO, TERCEIRO QUADRO DO OBJETO MOEDA



FONTE: Autoria própria (2018).

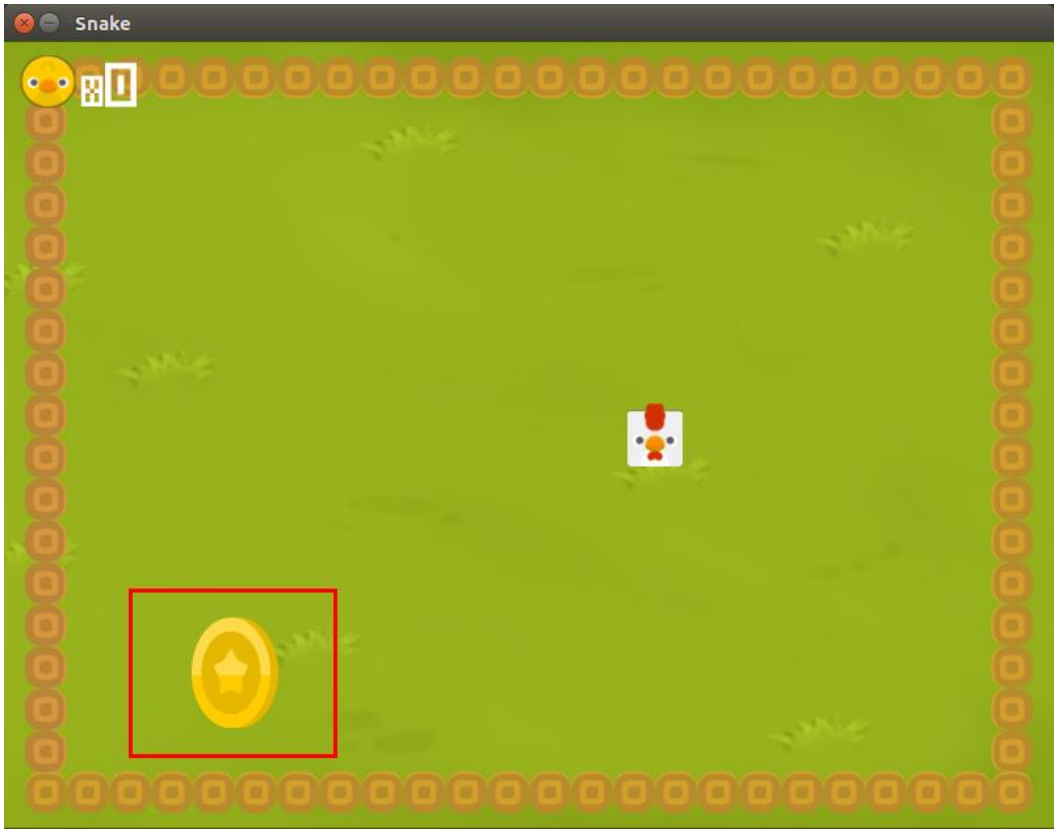
FIGURA 29 – APLICAÇÃO, QUARTO QUADRO DO OBJETO MOEDA

FONTE: Autoria própria (2018).

FIGURA 30 – APLICAÇÃO, QUINTO QUADRO DO OBJETO MOEDA

FONTE: Autoria própria (2018).

FIGURA 31 – APLICAÇÃO, SEXTO QUADRO DO OBJETO MOEDA



FONTE: Autoria própria (2018).

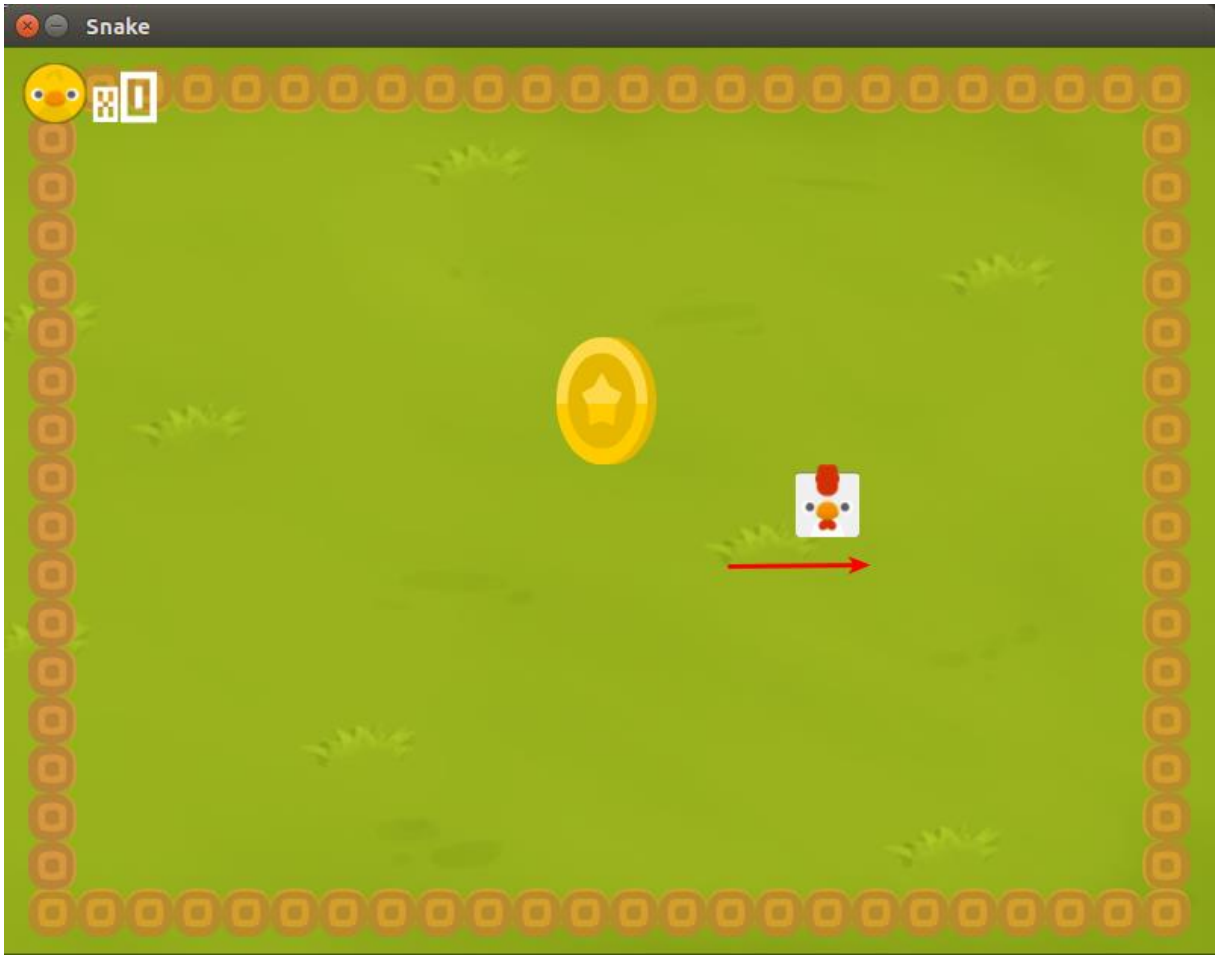
Por fim a função que diz respeito ao movimento do objeto dentro do jogo. Na Figura 32 é representada a função de movimento, passando como parâmetro o objeto, a direção, o vetor a ser alterado e a quantidade de pixels (podendo esta ser opcional caso deseje movimentar apenas um). O resultado desta função se encontra na Figura 33.

FIGURA 32 – CÓDIGO, MOVIMENTAÇÃO DE OBJETO

```
move(galinha, 48, COLUNA, DIREITA);
```

FONTE: Autoria própria (2018).

FIGURA 33 – APLICAÇÃO, MOVIMENTO DO OBJETO GALINHA



FONTE: Autoria própria (2018).

5 CONSIDERAÇÕES FINAIS

Este documento apresentou o desenvolvimento do *framework* B.E.A.G.L.E. O projeto começou com o objetivo de aprimorar o ensino de programação de computadores para estudantes iniciantes. Para isso, foi utilizado o formato de *framework,* que permite ao aluno o desenvolvimento de jogos com uma visão mais clara perante as atividades que desenvolve.

O *framework* em si foi desenvolvido no *software Processing* que tem como característica a flexibilidade, facilitando ainda mais o aprendizado e pelo fato de ser codificado em Java permite ao aluno um contato com uma linguagem amplamente utilizada no mercado. Contudo, o estudante tem a possibilidade de exercitar os conceitos de lógica, física, matemática e etc, sem a necessidade de possuir um conhecimento aprofundado na área de programação.

Os *frameworks* possuem diversas classificações e o B.E.A.G.L.E. se encaixa na classificação caixa cinza, que mescla tanto as características de caixa branca quanto caixa preta permitindo ao *framework* uma maior flexibilidade em seu desenvolvimento. Já entre as metodologias apresentadas o *framework* B.E.A.G.L.E. seguiu a metodologia de Henninger, que com o uso de hipertextos facilitou a reutilização de funcionalidades já desenvolvidas e facilitou o desenvolvimento de novas, além de seguir os conceitos de orientação a objetos que facilita ainda mais aprendizado do aluno.

Os requisitos foram embasados em funcionalidades e, por se tratar de um *framework,* a equipe decidiu por coletar os requisitos a partir dos casos de uso, os quais proporcionam uma coleta de requisitos mais precisa. Para a gerência de tarefas e também de tempo no desenvolvimento de cada funcionalidade foi utilizado o diagrama de *Gantt* que permitiu que a equipe controlasse de forma precisa o tempo gasto em cada tarefa.

Para realizar o elemento de hipertexto foi utilizado o diagrama UML, a fim de representar os atores que utilizam o *framework* e suas possíveis ações. Também foi utilizado o diagrama de classes, o qual converge com a metodologia de Henninger, a fim de demonstrar um embasamento técnico. E, por fim, para de demonstrar todas as características do *framework* foi desenvolvido o jogo “Chuva de meteoros - Um jogo B.E.A.G.L.E”, que demonstra todas as funcionalidades disponíveis do *framework*.

Para finalizar, resta responder a questão levantada no capítulo 1: “Como o desenvolvimento de jogos pode facilitar o aprendizado de lógica e programação de computadores?”. Bem, os jogos digitais conseguem atrair facilmente a atenção, principalmente dos mais jovens. Logo, uma ferramenta como o *framework* B.E.A.G.L.E. que, além de permitir a fixação dos conteúdos aprendidos em sala de aula, transmite também uma extensão de conhecimento da lógica de programação de forma mais simples, motivando ainda mais os alunos a serem produtores de seus próprios jogos.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Os trabalhos futuros doravante poderão dar continuidade na pesquisa e criar melhorias para o *framework* desenvolvido:

- A criação de uma interface de desenvolvimento seja usando bloco de códigos ou outra maneira para criar maior acessibilidade.
- Adaptar a interface e o *framework* para se tornar possível a utilização em dispositivos móveis ou uma plataforma web
- Realizar maior investimento em teoria pedagógica dos resultados do uso do *framework* como objeto de ensino.

REFERÊNCIAS

ADAMS, Douglas Noël, **The Long Dark Tea-Time of the Soul**, 1988, ISBN: 1476783004.

ALVES, João Amaro Barcelos, **GERÊNCIA DE PROJETOS: riscos e desafios na administração dos projetos nas organizações**, 2006. Disponível em: <http://bd.centro.iff.edu.br/jspui/handle/123456789/235>. Acesso em 23 de Novembro de 2018.

ALECRIM, Emerson. **Cluster: conceito e características**. Disponível em: <https://www.infowester.com/cluster.php />
Acesso em: 09 de outubro de 2018.

ANDRADE, Vinícius Camargo. **Classificação de Framework - Grupo de Pesquisa em engenharia de software**. Disponível em: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=2ahUKEwjj8OSYuubeAhWFHZAKHSW-AR8QFjAHegQIBxAC&url=http%3A%2F%2Fgpes.pg.utfpr.edu.br%2Fgpes%2Farquivos%2Fmaterial%2Fengenharia%2Fcontexto%2Frelatorios%2F2009%2FC.ANDRADE.2009.B.PDF&usg=AOvVaw0mkeb56siGTcgKApW7T6g_>
Acesso em: 21 de novembro de 2018.

AZEVEDO Junior, Delmir Peixoto de, CAMPOS, Renato de, **Definição de requisitos de software baseada numa arquitetura de modelagem de negócios**, 2008. Disponível em: <https://repositorio.unesp.br/handle/11449/29171>. Acesso em 23 de Novembro de 2018.

BARASSUOL, J. B., CHICON, MOZZAQUATRO, P. M., **O Desenvolvimento Do Raciocínio Lógico Através Da Engine Construct 2**, 2017. Disponível em: <https://home.unicruz.edu.br/seminario/anais/anais-2017>. Acesso em 21 de Novembro de 2018.

BATTAIOLA, A. L. Jogos por computador: Histórico, relevância tecnológica e mercadológica, tendências e técnicas de implementação. **Anais do XIX Jornada de Atualização em Informática**, p. 83–122, 2000.

BOOCH, Grady, RUMBAUGH, James, JACOBSON, Ivar, **The Unified Modeling Language User Guide**, 1998, ISBN: 0-201-57168-4.

CODESPEAKLABS. **What's the best first coding platform for young coders? Code.org vs Scratch**. Disponível em: <http://www.codespeaklabs.com/blog/what-s-the-best-first-coding-platform-for-young-coders-code-org > Acesso em : 10 de novembro de 2018.

Construct-2, Disponível em: <https://construct-2.br.uptodown.com/windows/>. Acesso em: 10 de dezembro de 2018.

FAYAD, Mohamed; Schmidt, Douglas C., **Frameworks de aplicações orientado a objetos**. Disponível em:
<https://projetos.inf.ufsc.br/arquivos_projetos/projeto_410/Resumo_artigo_2.pdf >
Acesso em : 08 de novembro de 2018.

GTK-PHP. Disponível em:
<http://www.php-gtk.com.br/home />Acesso em : 20 de novembro de 2018.

GASPAROTTO, Henrique, **Unity 3D: Introdução ao desenvolvimento de games**. Disponível em:
<https://www.devmedia.com.br/unity-3d-introducao-ao-desenvolvimento-de-games/30653>. Acesso em : 24 de outubro de 2018.

GERBRAN, Maurício, **Tecnologias Educacionais**, IESDE BRASIL SA, 2009, ISBN: 8538707396, 9788538707394.

GUEDES, GILLEANES T. A., **UML 2 - Guia Prático. 2a. Edição**. Novatec. 2014.

GAMMA, Erich, HELM, Richard, VLISSIDES, John, JOHNSON, Ralph. **Design Patterns: Elements of Reusable Object-Oriented Software**, Pearson Education,1994.

HAGUIWARA, E. V., OLIVEIRA, F. B., RODRIGUES, L., JOPPERT., L. S. S., JESUS, A., KUTZKE, A. R., **Oficina: Raciocinando Com O Computador**. Disponível em: <https://acervodigital.ufpr.br/handle/1884/57001>. Acesso em 18 de Novembro de 2018.

Java, Disponível em: <https://www.java.com/pt_BR/about/>. Acesso em 20 de Novembro de 2018.

JONES, Christopher V., **The Three-Dimensional Gantt Chart. Operations Research** 36(6):891-903, 1988. Disponível em: <
https://doi.org/10.1287/opre.36.6.891>. Acesso em 24 de Novembro de 2018.

JULL, J., **Half-Real: Video Games between Real Rules and Fictional Worlds**, The MIT Press, 2005, ISBN: 0262101106

JÚNIOR, J. C. R. P., RAPKIEWICZ, C. E., DELGADO, Carla, XEXEO, J. A. M., **Ensino de Algoritmos e Programação: Uma Experiência no Nível Médio**. Disponível em: <http://jacarepagua.dcc.ufrj.br/~ladybug/artigos/PereiraJr.pdf>. Acesso em 17 de Novembro de 2018.

JOHNSON, Ralph, FOOTE, Brian, **Designing Reusable Classes**. Disponível em:
<http://www.laputan.org/drc.html />
Acesso em : 09 de outubro de 2018.

LUCCHES, Fabiano; RIBEIRO, Bruno. **Conceituação de Jogos Digitais**. Disponível em:
<http://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g3.pdf >
Acesso em: 20 de outubro de 2018.

MANIERO, Antonio. **Qual é a diferença de API, biblioteca e Framework?.** Disponível em: <https://pt.stackoverflow.com/questions/17501/qual-%C3%A9-a-diferen%C3%A7a-de-api-biblioteca-e-framework />Acesso em : 25 de outubro de 2018.

MEDEIROS, T. J., **Um Framework para criação de jogos voltados para o ensino de lógica de programação**, 2014.

MELO, Ivana. **O Uso De Jogos Eletrônicos Como Ferramenta De Ensino: Um Estudo Da Suíte De Jogos Gcompris**. Disponível em: <http://www2.unifap.br/midias/files/2016/04/O-uso-de-jogos-eletr%C3%B4nicos-como-ferramentas-educacional-complementar-IVANA-RALIENE-PAIX%C3%83O-DE-MELO.pdf>Acesso em : 13 de outubro de 2018.

PICANÇO, Wollace De Souza, **FdRobô: Um Framework Didático Para Auxiliar O Ensino De Linguagem De Programação Adaptada Ao Método De Aprendizagem Cooperativa E Competitiva**, Amazonas, 2016.

PROCESSING. Disponível em: <https://processing.org/overview />. Acesso em: 11 de outubro de 2018.

QTDESIGNER. Disponível em: <http://doc.qt.io/qt-5/qtdesigner-manual.html />. Acesso em: 11 de novembro de 2018.

ROGERS, Gregory. **Framework-Based Software Development in C++**, Prentice Hall,1997.

SCHUYTEMA.P. **Design de games: uma abordagem prática**. São Paulo: Cengag Learning, 2008. 447 p.

SILVA, Ricardo Pereira E., **Suporte ao desenvolvimento e uso de frameworks e componentes. Porto Alegre: Universidade Federal do Rio Grande do Sul**, 2000. 37p .262 p. (Tese Doutorado em Ciência da Computação).

SOARES, Michel dos Santos, **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**, 2004. Disponível em: <http://www.dcc.ufla.br/infocomp/index.php/INFOCOMP/article/view/68>. Acesso em 21 de Novembro de 2018.

TURINE, M. A. S., MASIERO, P. C., **Especificação De Requisitos: Uma Introdução**, 1996. Disponível em: <https://sistemas.riopomba.ifsudestemg.edu.br/dcc/materiais/2109177910_aula%20extra%204%20engenharia_de_requisitos.pdf>. Acesso em 25 de Novembro de 2018.

Unity, Disponível em: <https://unity.com/madewith/adam />. Acesso em: 10 de Dezembro de 2018.

ZANETTE. Alysson. **Framework x Biblioteca x API. Entenda as diferenças!** Disponível em:
< https://becode.com.br/framework-biblioteca-api-entenda-as-diferencas/ >
Acesso em : 25 de outubro de 2018.

ZIMMERMANN, William, Disponível em:
<https://www.williamzimmermann.com.br/category/desenvolvimento/php-mysql/laravel/ >. Acesso em 01 de Novembro de 2018.

APÊNDICE 1 – DESCRIÇÃO DE CASO DE USO

QUADRO 6 – CASO DE USO “UC01 - CRIAR OBJETO”

CASO DE USO	UC01 - Criar Objeto
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Realiza a criação de um novo objeto.
PRÉ-CONDIÇÕES	1. Utilizar a biblioteca B.E.A.G.L.E.
FLUXO PRINCIPAL	1. Sistema desenvolvido pelo usuário requisita o método “criarObjeto” passando como parâmetro o id “OBJETO” e as seguintes propriedades: altura, largura, posicionamento x, posicionamento y e posicionamento z, sendo as 3 últimas opcionais. FA01 FA02 2. O framework B.E.A.G.L.E. cria uma nova instância do objeto. 3. O framework B.E.A.G.L.E. salva o objeto em uma variável de lista. 4. O framework B.E.A.G.L.E. retorna o objeto para o usuário utilizar. 5. O caso de uso é encerrado.PC01
FLUXO ALTERNATIVO	FA01 1. Se objeto a ser criado for do tipo “OBJETO_ILUSTRADO” se inicia o caso de uso “UC02 - Criar objeto ilustrado”. FA02 2. Se objeto a ser criado for do tipo “OBJETO_ANIMADO” se inicia o caso de uso “UC02 - Criar objeto animado”.
EXCEÇÕES	Os parâmetros não condizem com os tipos dos atributos.
PÓS-CONDIÇÕES	O objeto é instanciado.

REGRA	Não se aplica.

FONTE: Autoria própria (2018).

QUADRO 7 – CASO DE USO “UC02 - CRIAR OBJETO ILUSTRADO”

CASO DE USO	UC02 - Criar objeto ilustrado
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Realiza a criação de um novo objeto ilustrado.
PRÉ-CONDIÇÕES	1. Utilizar a biblioteca B.E.A.G.L.E.
FLUXO PRINCIPAL	1. Sistema desenvolvido pelo usuário requisita o método “criarObjeto” passando como parâmetro id ”OBJETO_ILUSTRADO” e as seguintes propriedades: altura, largura, posicionamento x, posicionamento y e posicionamento z, sendo as 3 últimas opcionais. 2. O framework B.E.A.G.L.E. cria uma nova instância do objeto. 3. O framework B.E.A.G.L.E. salva o objeto em uma variável de lista. 4. O framework B.E.A.G.L.E. retorna o objeto para o usuário utilizar. 5. O caso de uso é encerrado.
FLUXO ALTERNATIVO	Não se aplica.
EXCEÇÕES	Os parâmetros não condizem com os tipos dos atributos.
PÓS-CONDIÇÕES	O objeto é instanciado.
REGRA	Não se aplica.

FONTE: Autoria própria (2018).

QUADRO 8 – CASO DE USO “UC03 - CRIAR OBJETO ANIMADO”

CASO DE USO	UC03 - Criar objeto animado
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Realiza a criação de um novo objeto ilustrado.
PRÉ-CONDIÇÕES	1. Utilizar a biblioteca B.E.A.G.L.E.
FLUXO PRINCIPAL	1. O Sistema desenvolvido pelo usuário requisita o método “criarObjeto” passando como parâmetro id ”OBJETO_ANIMADO” e as seguintes propriedades: altura, largura, posicionamento x, posicionamento y e posicionamento z, sendo as 3 últimas opcionais. 2. O framework B.E.A.G.L.E. cria uma nova instância do objeto. 3. O framework B.E.A.G.L.E. salva o objeto em uma variável de lista. 4. O framework B.E.A.G.L.E. retorna o objeto para o usuário utilizar. 5. O caso de uso é encerrado.
FLUXO ALTERNATIVO	Não se aplica.
EXCEÇÕES	Os parâmetros não condizem com os tipos dos atributos.
PÓS-CONDIÇÕES	O objeto é instanciado.
REGRA	Não se aplica.

FONTE: Autoria própria (2018).

QUADRO 9 – CASO DE USO “UC04 - DEFINIR EM DESENVOLVIMENTO”

CASO DE USO	UC04 - Definir em desenvolvimento.
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Definir *flag* que determina se o programa está em fase de desenvolvimento, apresentando ao desenvolvedor uma visão do espaço ocupado pelo objeto.
PRÉ-CONDIÇÕES	1. Usar a biblioteca B.E.A.G.L.E. no sistema desenvolvido.
FLUXO PRINCIPAL	1. O sistema desenvolvido chama o método “defineEmDesenvolvimento” passando uma variável booleana com o valor “true”; FA01 2. O framework B.E.A.G.L.E. define o valor do atributo “emDesenvolvimento” com o valor “true”, para todos os objetos já criados; 3. O caso de uso se encerra. PC01
FLUXO ALTERNATIVO	1. O sistema desenvolvido chama o método “defineEmDesenvolvimento” passando uma variável boolean com o valor false 2. O framework B.E.A.G.L.E. define o valor do atributo “emDesenvolvimento” com o valor “true”, para todos os objetos já criados; 3. O caso de uso se encerra. PC02
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01. 1. Ao imprimir um objeto, será apresentado o desenho da área de colisão que a ele pertence. PC02.

	1. Ao imprimir um objeto, não será apresentado o desenho da caixa de colisão que a ele pertence.
REGRA	Deve ser enviado uma variável booleana ao método que realiza esta função.

FONTE: Autoria própria (2018).

QUADRO 10 – CASO DE USO “UC05 - VERIFICAR COLISÃO”

CASO DE USO	UC05 - Verificar Colisão
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Verifica se houve colisão entre objetos.
PRÉ-CONDIÇÕES	1. Utilizar a biblioteca B.E.A.G.L.E. 2. Possuir dois objetos criados.
FLUXO PRINCIPAL	1. O sistema desenvolvido pelo usuário chama o método “estaColidindo”, passando como parâmetros duas variáveis Objeto que o framework validará. 2. O framework B.E.A.G.L.E. verifica que a área relativa ao tamanho do primeiro objeto (altura x largura) está parcialmente (ou totalmente) sobrescrevendo a área do segundo objeto recebido por parâmetro. FA01 3. O framework B.E.A.G.L.E. retorna ao sistema desenvolvido pelo usuário uma variável booleana com o valor (true), informando que os objetos estão colidindo. 4. O caso de uso se encerra.
FLUXO ALTERNATIVO	FA01 1. O framework B.E.A.G.L.E. verifica que a área relativa ao tamanho do primeiro objeto não sobrescreve a área do segundo objeto recebido por parâmetro. 2. O framework B.E.A.G.L.E. retorna ao sistema desenvolvido pelo usuário uma variável booleana com o

	valor (falso), informando que os objetos não estão colidindo. 3. O caso de uso se encerra.
EXCEÇÕES	Não se Aplica.
PÓS-CONDIÇÕES	O usuário consegue verificar se ambos os objetos estão colidindo.
REGRA	Devem ser passados por parâmetro duas variáveis de objetos já criadas.

FONTE: Autoria própria (2018).

QUADRO 11 – CASO DE USO “UC06 - VERIFICAR RESTITUIÇÃO”

CASO DE USO	UC06 - Verificar Restituição
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Retorna o resultado de colisão entre dois objetos.
PRÉ-CONDIÇÕES	1. Utilizar a biblioteca B.E.A.G.L.E. 2. Possuir dois objetos criados. 3. A verificação de colisão ter sido realizada. 4. Os objetos devem estar colidindo
FLUXO PRINCIPAL	1. O sistema solicita o método "resolveColisao" passando os dois objetos que entraram em colisão 2. O framework B.E.A.G.L.E. calcula novas posições finais para os dois objetos com base em suas velocidades no momento da colisão 3. O framework movimenta o objeto na tela em direção as posições calculadas 4. O caso de uso se encerra. PC01.
FLUXO ALTERNATIVO	Não se Aplica.

EXCEÇÕES	Não se Aplica
PÓS-CONDIÇÕES	PC01 1. O framework posiciona os objetos conforme resposta da colisão.
REGRA	Sistema deve retornar o resultado da colisão

FONTE: Autoria própria (2018).

QUADRO 12 – CASO DE USO “UC07 - IMPRIMIR OBJETO”

CASO DE USO	UC07 - Imprimir Objeto
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Imprime o objeto em tela.
PRÉ-CONDIÇÕES	1. Utilizar a biblioteca B.E.A.G.L.E. 2. Possuir um objeto criado.
FLUXO PRINCIPAL	1. Sistema solicita o método “imprime” passando como parâmetro um objeto ou um objeto ilustrado a ser impresso. FA01 2. O framework B.E.A.G.L.E. desenha na tela o objeto passado por parâmetro, respeitando o ângulo em que o objeto se encontra. 3. O caso de uso é encerrado. PC01
FLUXO ALTERNATIVO	FA01 1. Caso o objeto seja do tipo “ObjetoAnimado” é iniciado o caso de uso “UC08 Imprimir objeto animado”.
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01 1. O objeto é exibido na tela.

REGRA	Devem ser passados por parâmetro uma variável de objeto já criada.

FONTE: Autoria própria (2018).

QUADRO 13 – CASO DE USO “UC08 - IMPRIMIR OBJETO ANIMADO”

CASO DE USO	UC08 - Imprimir objeto animado
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Imprime uma animação de um o objeto do tipo “ObjetoAnimado” em tela.
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto do tipo “ObjetoAnimado” já criado.
FLUXO PRINCIPAL	1. Sistema solicita o método “imprime” passando como parâmetro o identificador da animação desejada e um objeto animado. 2. O framework B.E.A.G.L.E. busca no objeto a animação relativa ao identificador. 3. O framework B.E.A.G.L.E. desenha na tela a animação relativa ao identificador, do objeto passado por parâmetro, respeitando o ângulo em que o objeto se encontra. FA01 4. O caso de uso é encerrado. PC01
FLUXO ALTERNATIVO	FA01 1. Caso não haja no objeto animado uma animação com o ID, finaliza a execução do caso de uso. PC02
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01 1. O objeto é exibido na tela. PC02

	1. O objeto não é exibido na tela.
REGRA	Devem ser passados por parâmetro uma variável de objeto já criada.

FONTE: Autoria própria (2018).

QUADRO 14 – CASO DE USO “UC09 - IMPRIMIR COLISÃO”

CASO DE USO	UC09 - Imprimir colisão
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Imprime a área de colisão de um objeto.
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado.
FLUXO PRINCIPAL	1. Sistema executa o método “colisaoImpressao” passando um objeto como parâmetro. 2. O framework B.E.A.G.L.E. imprime um retângulo rosa, com as proporções de largura e altura, indicando a área de colisão daquele objeto, respeitando o ângulo em que o objeto se encontra. 3. O caso de uso é encerrado. PC01
FLUXO ALTERNATIVO	Não se aplica.
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01 1. A colisão é exibida.
REGRA	Devem ser passados por parâmetro uma variável de objeto já criada.

FONTE: Autoria própria (2018).

QUADRO 15 – CASO DE USO “UC10 - SEGUIR MOUSE”

CASO DE USO	UC10 - Seguir Mouse
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Sistema faz objeto se movimentar em direção à posição atual, relativa do mouse
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado.
FLUXO PRINCIPAL	1. O sistema solicita o método “segueMouse”, passando a velocidade do movimento e o objeto que será movimentado. 2. O framework B.E.A.G.L.E. recupera a posição do mouse em tela. 3. O framework B.E.A.G.L.E. movimenta o objeto na tela, em direção ao valor recuperado do mouse, na velocidade definida passado por parâmetro 4. O caso de uso de encerra. PC01
FLUXO ALTERNATIVO	Não se aplica
EXCEÇÕES	Não se aplica
PÓS-CONDIÇÕES	PC01 1. O framework movimenta o objeto em direção ao mouse.
REGRA	Devem ser passados por parâmetro uma variável de objeto já criada.

FONTE: Autoria própria (2018).

QUADRO 16 – CASO DE USO “UC11 - ROTACIONAR AO MOUSE“

CASO DE USO	UC11 - Rotacionar ao Mouse
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Sistema rotaciona o objeto de acordo com a posição do mouse
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado.
FLUXO PRINCIPAL	1. O sistema solicita o método “rotacionarParaMouse”, passando um objeto como parâmetro. 2. O framework B.E.A.G.L.E. recupera a posição relativa do mouse na tela. 3. O framework B.E.A.G.L.E. gira o objeto em seu eixo, mantendo alinhado 90º com a posição relativa do mouse. 4. O ângulo de rotação do objeto é redefinido. 5. O caso de uso de encerra.PC01
FLUXO ALTERNATIVO	Não se Aplica
EXCEÇÕES	Não se Aplica
PÓS-CONDIÇÕES	PC01 1. Sistema gira o objeto em relação ao mouse.
REGRA	Devem ser passados por parâmetro uma variável de objeto já criada.

FONTE: Autoria própria (2018).

QUADRO 17 – CASO DE USO “UC12 - GIRAR OBJETO”

CASO DE USO	UC12 - Girar objeto
ATOR	Sistema desenvolvido pelo usuário.
DESCRIÇÃO	Gira o objeto em 1º, sentido anti-horário
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado.
FLUXO PRINCIPAL	1. O sistema solicita o método “gira”, passando o objeto quem deseja rotacionar. 2. O framework B.E.A.G.L.E. rotaciona o objeto ao redor se seu centro em 1º sentido anti-horário. 3. O caso de uso se encerra.
FLUXO ALTERNATIVO	Não se aplica.
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	Não se aplica.
REGRA	Devem ser passados por parâmetro uma variável de objeto já criada.

FONTE: Autoria própria (2018).

QUADRO 18 – CASO DE USO “UC13 - MOVIMENTAR OBJETO”

CASO DE USO	UC13 - Movimentar objeto
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	O sistema movimenta o objeto de acordo com as iterações realizadas
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado.

FLUXO PRINCIPAL	1. O sistema chama o método move, passando como parâmetro o objeto, a direção (INCREMENTA, DECREMENTA), o vetor a ser alterado (LINHA, COLUNA, PROFUNDIDADE) e a quantidade de pixels (podendo esta ser opcional caso deseje movimentar apenas um. 2. O framework B.E.A.G.L.E. interpreta a passagem de parâmetro por meio de variáveis globais e movimenta o objeto na tela, na direção desejada. 3. O caso de uso de encerra.
FLUXO ALTERNATIVO	Não se aplica.
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01 1. O objeto é movimentado.
REGRA	Não se aplica.

FONTE: Autoria própria (2018).

QUADRO 19 – CASO DE USO “UC14 - ADICIONAR IMAGEM”

CASO DE USO	UC14 - Adicionar imagem
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Solicita a adição de imagem a um objeto ilustrado
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto ilustrado já criado.
FLUXO PRINCIPAL	1. O sistema solicita o método “adicionaImagem” passando como parâmetro o nome da imagem e o objeto ilustrado que a animação pertencerá.

	2. O framework encontra a imagem pelo atributo recebidos. FE01 3. O framework monta uma nova instância de animação e a associa ao objeto animado passado por parâmetro. 4. O caso de uso é encerrado.PC01
FLUXO ALTERNATIVO	Não se aplica
EXCEÇÕES	FE01 1. A imagem não foi encontrada pelo framework 2. O caso de uso se encerra.
PÓS-CONDIÇÕES	PC01 1. Uma nova imagem é adicionada ao objeto
REGRA	O objeto deve ser do tipo Objeto ilustrado

FONTE: Autoria própria (2018).

QUADRO 20 – CASO DE USO “UC15 - ADICIONAR ANIMAÇÃO”

CASO DE USO	UC15 - Adicionar animação
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Solicita a adição de animação a um objeto animado
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto animado já criado. 3. Possuir imagens sequenciais, com o mesmo sufixo em seu nome, de mesma extensão, e que possua uma sequência numérica de dois caracteres. Exemplo: “imagem01.jpg”, “imagem02.jpg”
FLUXO PRINCIPAL	1. O sistema solicita o método “adicionaAnimacao” passando como parâmetro um identificador para a animação, o prefixo do nome da imagem, a quantidade de

	imagens que irão compor a animação, a extensão do tipo de arquivo (sendo opcional para buscas de arquivos ".png") e o objeto animado que a animação pertencerá. 2. O framework encontra as imagens pelos atributos recebidos. FE01 3. O framework monta uma nova instância de animação e a associa ao objeto animado passado por parâmetro. 4. O caso de uso é encerrado. PC01
FLUXO ALTERNATIVO	Não se aplica
EXCEÇÕES	FE01 1. Alguma imagem não foi encontrada pelo framework 2. O caso de uso se encerra.
PÓS-CONDIÇÕES	PC01 1. Uma nova animação é adicionada ao objeto
REGRA	O objeto deve ser do tipo Objeto animado

FONTE: Autoria própria (2018).

QUADRO 21 – CASO DE USO "UC16 - ADICIONAR ÁUDIO"

CASO DE USO	UC16 - Adicionar áudio
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Realiza a adição de áudio
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Ter um arquivo de áudio no formato de .mp3 ou .wav na mesma pasta que o sistema do usuário, ou dentro de uma pasta nomeada "data". 3. Ter a biblioteca "processing.sound" instalada em seu ambiente Processing.

FLUXO PRINCIPAL	1. O sistema solicita a adição de um novo áudio, passando como parâmetro o nome do arquivo. 2. O framework busca por um arquivo de áudio com o nome passado por parâmetro. 3. Após encontrar o arquivo relativo ao nome, o framework instancia um novo objeto do tipo Audio. FA01 4. O framework B.E.A.G.L.E. adiciona esta mesma instância o arquivo em uma lista, referenciando o mesmo pelo nome passado por parâmetro, como uma chave identificadora. 5. O caso de uso se encerra
FLUXO ALTERNATIVO	FA01 1. Caso não seja possível encontrar um arquivo com o mesmo nome referenciado, o caso de uso se encerra.
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01 1. Uma nova faixa de áudio é adicionada à lista de áudios.
REGRA	Não se aplica

FONTE: Autoria própria (2018).

QUADRO 22 – CASO DE USO “UC17 - TOCAR ÁUDIO”

CASO DE USO	UC17 - Tocar áudio
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Solicita a reprodução de áudio.
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado. 3. Possuir áudios já adicionados na ferramenta.

FLUXO PRINCIPAL	1. O sistema requisita o método “tocarAudio” passando como parâmetro o nome do arquivo de áudio. 2. O sistema realiza a reprodução do áudio. FA01 3. O caso de uso se encerra. PC01
FLUXO ALTERNATIVO	Se nenhum arquivo de áudio for passado como parâmetro ou o mesmo ser inexistente nenhum áudio será reproduzido
EXCEÇÕES	Não se aplica.
PÓS-CONDIÇÕES	PC01 1. Ao realizar uma ação o áudio será reproduzido.
REGRA	Não se aplica

FONTE: Autoria própria (2018).

QUADRO 23 – CASO DE USO “UC18 - TOCAR ÁUDIO CONTÍNUO”

CASO DE USO	UC18 - Tocar áudio contínuo
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Solicita a reprodução de áudio de forma contínua.
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado. 3. Possuir áudios já adicionados na ferramenta.
FLUXO PRINCIPAL	1. O sistema requisita o método “tocarAudio” passando como parâmetro o nome do arquivo de áudio. 2. O framework B.E.A.G.L.E. encontra o áudio condizente ao nome do arquivo passado por parâmetro, nos arquivos do programa. FE01 3. O sistema realiza a reprodução do áudio. 4. O caso de uso se encerrado. PC01

FLUXO ALTERNATIVO	Não se aplica
EXCEÇÕES	FE01 1. Se nenhum arquivo de áudio for passado como parâmetro ou o mesmo ser inexistente, nenhum áudio será reproduzido.
PÓS-CONDIÇÕES	PC01 1. O áudio será reproduzido de forma contínua.
REGRA	Não se aplica

FONTE: Autoria própria (2018).

QUADRO 24 – CASO DE USO “UC19 - PARAR AUDIO”

CASO DE USO	UC19 - Parar Áudio
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Solicita a parada da reprodução de áudio.
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado. 3. Possuir áudio em reprodução.
FLUXO PRINCIPAL	1. O sistema requisita o método “pararAudio” passando como parâmetro o nome do arquivo de áudio. 2. O framework B.E.A.G.L.E. encontra o áudio condizente ao nome do arquivo passado por parâmetro, nos arquivos do programa. FE01 3. O sistema para a reprodução do áudio. 4. O caso de uso se encerrado. PC01
FLUXO ALTERNATIVO	FA01 1. Caso não encontre um áudio para o arquivo requisitado, o caso de uso se encerra.

EXCEÇÕES	Não se aplica
PÓS-CONDIÇÕES	PC01 1. O áudio que estava em reprodução é parado.
REGRA	Não se aplica.

FONTE: Autoria própria (2018).

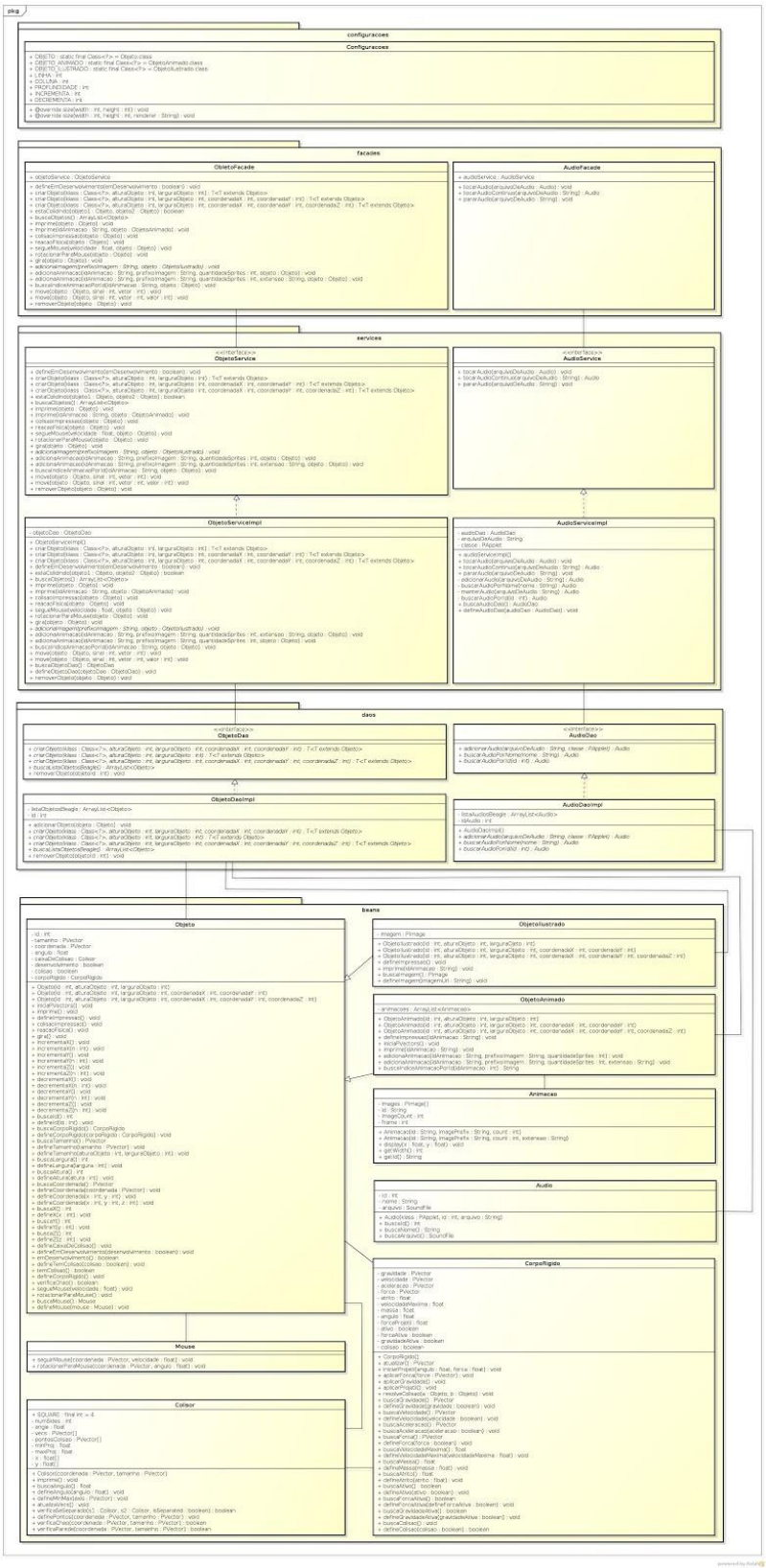
QUADRO 25 – CASO DE USO “UC20 - REMOVER OBJETO”

CASO DE USO	UC20 - Remover objeto
ATOR	Sistema desenvolvido pelo usuário
DESCRIÇÃO	Remover um objeto criado da lista de controle
PRÉ-CONDIÇÕES	1. Utilizar a ferramenta B.E.A.G.L.E. 2. Possuir um objeto já criado.
FLUXO PRINCIPAL	1. O ator realiza a chamada do método “removerObjeto”, passando um objeto como parâmetro; 2. O sistema busca o índice do objeto na lista, pelo id do objeto; FA01 3. O sistema remove o objeto; 4. Encerra o caso de uso.
FLUXO ALTERNATIVO	FA01 1. Caso não encontre o objeto requisitado ou o mesmo não tenha sido instanciado, o caso de uso se encerra.
EXCEÇÕES	Não se aplica
PÓS-CONDIÇÕES	PC01 1. O objeto é removido da lista de objetos.
REGRA	Não se aplica.

FONTE: Autoria própria (2018).

APÊNDICE 2 – DIAGRAMA DE CLASSES

FIGURA 34 – DIAGRAMA DE CLASSES



FONTE: Autoria própria (2018).